

SLOTH TUTORIAL · THE BASICS

SLOTH input files in a nutshell



Four ideas to understand a SLOTH input file

The input file is a program

No text input file: a compiled main.cpp that assembles SLOTH objects.

One equation = one operator + integrators

TransientOPE or SteadyOPE depending on the type of simulation,

+ named integrators that carry the variational formulation of the problem.

Physics is specified analytically

The energy functional and the kinetic parameters are SLOTH Coefficient objects, written in a JSON file.

A multiphysics scheme built by nesting

`TimeDiscretization > { Coupling > { Problem } }`

Adding a physics = adding a Problem.

An input file (main.cpp) in 4 main parts

1

Headers and parallelism

includes, MPI and Hypr, namespace and aliases, footer

2

Spatial discretization

mesh (SPA), boundary conditions (BCS)

3

Multiphysics coupling

Coefficients, Variables, Operators + Integrators, Post-processing, Problems, Coupling

4

Time marching + solve

Opening and closing a simulation

1

Headers and
parallelism

2

Spatial
discretization

3

Multiphysics
coupling

4

Time
discretization

```
// — header —
#include "Sloth/sloth.hpp"      // all of SLOTH
#include "Sloth/tests.hpp"      // test utilities
#include "MyCoefficients.hpp"   // your generated .hpp
int main(int argc, char* argv[]) {
    setVerbosity(Verbosity::Normal); // output level
    mfem::Mpi::Init(argc, argv);      // MPI
    mfem::Hypre::Init();               // Hypre solvers
    Profiling::getInstance().enable();
    using namespace Sloth2D;          // dim., aliases

    // ... parts 2, 3 and 4 ...

// — footer —
    Profiling::getInstance().print(); // timings
    MPI_Finalize();
    return 0;
}
```

using namespace Sloth2D;

(or Sloth1D, Sloth3D) gives access to the
main aliases:

SPA	→ spatial discretization
SPAS	→ vector of SPA
BCS	→ boundary conditions
VAR	→ one variable
VARs	→ a set of VAR
PST	→ post-processing
TransientOPE	→ an operator type
SteadyOPE	
TransientPB	→ a problem type
SteadyPB	

The mesh: a SPA object

1

Headers and
parallelism

2

Spatial
discretization

3

Multiphysics
coupling

4

Time
discretization

One call, four arguments (minimum)

```
SPA spatial("InlineSquareWithQuadrangles", 1, 0, std::make_tuple(200, 200, 50., 50.));
```

- ① **mesh type** · GMSH file or MFEM “inline” mesh
- ② **element order** · 1 = linear
- ③ **uniform refinements** · 0 = none
- ④ **dimensions** · cells and lengths per direction

InlineLineWithSegments · 1D

InlineSquareWithQuadrangles · Triangles · 2D

...Tetraedres · Hexaedres · 3D

Optional

Periodic

..., translations)

AMR · non-conforming

..., true)

GMSH file (.msh ASCII v2)

SPA("GMSH", 1, 0,
"mesh.msh")

Boundary conditions: a BCS object

1

Headers and
parallelism

2

Spatial
discretization

3

Multiphysics
coupling

4

Time
discretization

One Boundary per edge

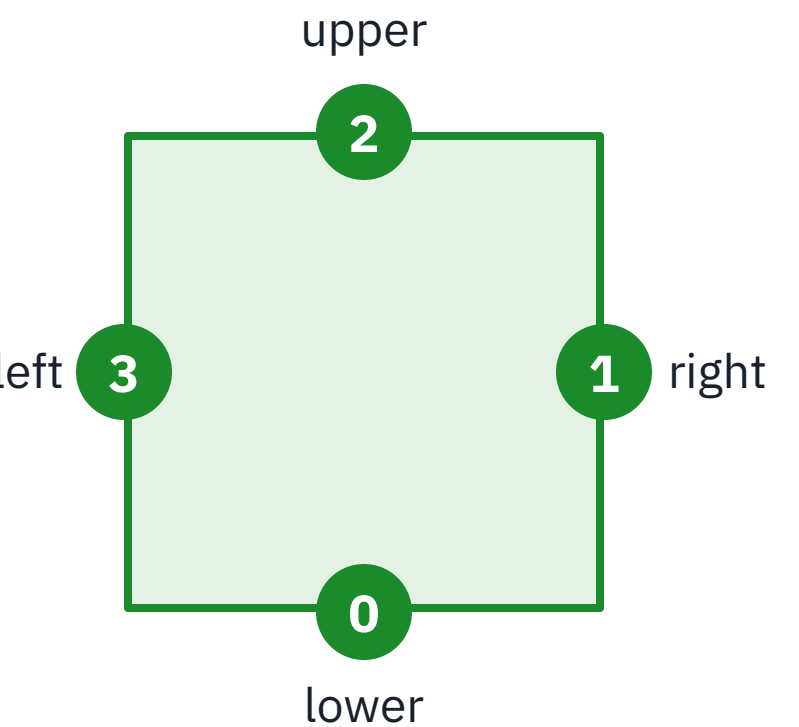
```
BCS bcs(&spatial, {  
  Boundary("lower", 0, "Neumann"),  
  Boundary("right", 1, "Dirichlet", 1.),  
  Boundary("upper", 2, "Neumann"),  
  Boundary("left", 3, "Dirichlet", 0.)});
```

Every edge

one Boundary per edge, no
exception

The right index

the name is free, the index is the
edge attribute in the mesh



Four types

Dirichlet (with its constant value),
Neumann, Robin, Periodic

Several fields on one mesh (AMR, ...)

1

Headers and
parallelism

2

Spatial
discretization

3

Multiphysics
coupling

4

Time
discretization

One mesh, several fields

```
SPA spatial_phi(..., true);  
SPA spatial_mu(  
    spatial_phi.get_mesh(), 1);
```

Required for AMR; orders are free.

N spaces in one call

```
SPAS spatials =  
    setSpatialDiscretization(  
        N, "InlineSquareWithQuadrangles",  
        ...);
```

Free with: deleteSpatialDiscretization.

N boundary conditions

```
auto bcs = setBoundaryConditions(  
    N, spatials, boundaries);
```

A single list: $bcs[i] \leftrightarrow spatials[i]$.

Non-uniform values

```
Coefficient bc(Glossary::Dirichlet, ...);  
bc.set_bdr_index_coef({1});
```

Function of x, y, t in a JSON file.

Mesh and boundary conditions, together

1

Headers and
parallelism

2

Spatial
discretization

3

Multiphysics
coupling

4

Time
discretization

Two fields, by hand φ and μ

```
// shared, non-conforming mesh
SPA spatial_phi(
    "InlineSquareWithQuadrangles",
    1, 0, std::make_tuple(200,200,50.,50.),
    true);
SPA spatial_mu(spatial_phi.get_mesh(), 1);
// one common list of edges
auto boundaries = {
    Boundary("lower", 0, "Neumann"),
    Boundary("right", 1, "Neumann"),
    Boundary("upper", 2, "Neumann"),
    Boundary("left", 3, "Neumann")};
// one BCS per variable
BCS bcs_phi(&spatial_phi, boundaries);
BCS bcs_mu(&spatial_mu, boundaries);
```

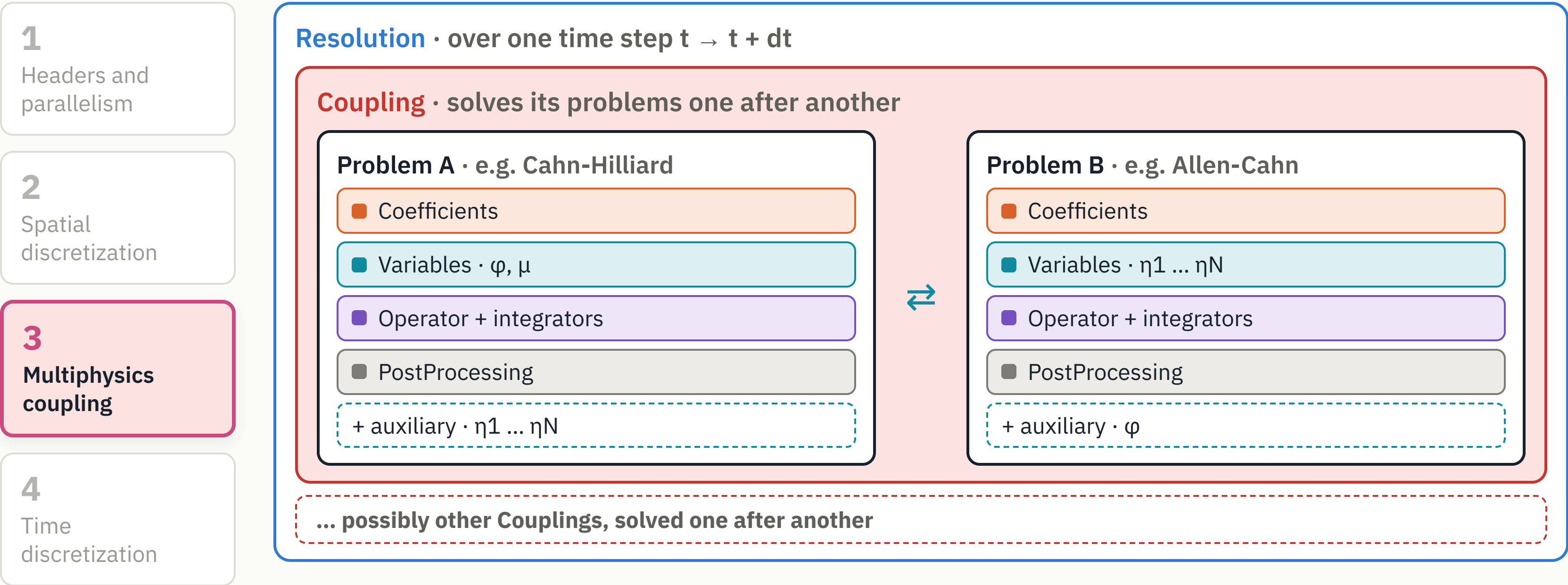
N fields, factory *going further*

```
const int N = 5; // 5 grains
// N spaces, one mesh
SPAS spatials = setSpatialDiscretization(
    N, "InlineSquareWithQuadrangles",
    1,0, std::make_tuple(200,200,50.,50.),
    true);
// same list of edges
auto bcs = setBoundaryConditions(
    N, spatials, boundaries);

// ... variables, problems, time ...

// in the footer
deleteSpatialDiscretization(spatials);
```

One time step: nested boxes



Order: **Coefficients** → **Variables** → **Operator** → PostProcessing → **Problem** → **Coupling** → **Resolution**

Physics: Coefficient objects

1

Headers and
parallelism

2

Spatial
discretization

3

Multiphysics
coupling

4

Time
discretization

```
// a constant
Coefficient capillary(Glossary::Capillary, lambda);
// a function provided by SLOTH
Coefficient w(Glossary::FreeEnergy, Scheme::Implicit, W(omega));
// a function written in coefficients.json, generated by Sympy
Coefficient f(Glossary::FreeEnergy, Scheme::Implicit, Gc());
// the collection given to the problem
Coefficients ch_coef(f, capillary, mobility, grad_energy);
```

Type

a Glossary quantity: **FreeEnergy**,
Mobility, **Capillary**, **GradEnergy**...

Scheme

Implicit, **Explicit** or **SemiImplicit**
(except **constants**)

Function

constant, **file**, **function**
(FunctionCoefficient from JSON) or
string (ExprTk)

Convention: in a function, the unknowns follow the order of the problem's **Variables** and then those of the auxiliary variables.

The unknowns: VAR and VARS

1

Headers and
parallelism

2

Spatial
discretization

3

Multiphysics
coupling

4

Time
discretization

```
auto phi = VAR(&spatial, bcs, "phi", Glossary::PhaseField, 2, phi_ic);
auto mu  = VAR(&spatial, bcs, "mu", Glossary::ChemicalPotential, 2, mu_ic);
auto ch_vars = VARS(phi, mu);    // the Cahn-Hilliard unknowns
```

SPA and BCS · the space and its boundary conditions (part 2)

name · used in the ParaView outputs

type · a Glossary quantity

storage · storage depth (2 in the example)

initial condition · constant, AnalyticalFunctions, std::function or restart file

Primary

solved by the problem: φ and μ for Cahn-Hilliard

Auxiliary

solved by another problem, only read;
passed as the problem's last argument

One physics = one operator + integrators

1

Headers and
parallelism

2

Spatial
discretization

3

Multiphysics
coupling

4

Time
discretization

$$\mathbf{F}(\mathbf{U}) = \mathbf{G}(\mathbf{U})$$

time derivative spatial operators

Solved by a unified Newton-Raphson (LBFGS also available).
Each term = an integrator providing residual and Jacobian.

TransientOPE · transient

```
TransientOPE(spatials,  
  {"CahnHilliard"},  
  scheme, "SplitTimeDerivative")
```

SteadyOPE · steady, $F = 0$

```
SteadyOPE(spatials,  
  {"AllenCahn"})
```

Time · F

TimeDerivative SplitTimeDerivative HeatTimeDerivative ...

Phase field · G

AllenCahn SplitAllenCahn CahnHilliard ...

Diffusion, heat · G

Fick MassFlux Fourier ...

Melting · G

MeltingConstant MeltingCalphad LatentHeat ...

Sources, advection · G

SourceTerm Advection SplitAdvection ...

📖 All integrators and their equations: [manual](#)

Post-processing (PST)

1

Headers and
parallelism

2

Spatial
discretization

3

**Multiphysics
coupling**

4

Time
discretization

```
auto pst_params = Parameters(  
    Parameter("main_folder_path", "Saves"),  
    Parameter("calculation_path", "CahnHilliard"),  
    Parameter("frequency", 100));  
auto ch_pst = PST(&spatial, pst_params);
```

Instead of **frequency** : **iterations_list** or **times_list** to choose the outputs.

The variables of the associated problem(s) are written in ParaView format (VTK).

Optional

no PST, no ParaView output for this problem

Shareable

one PST for several problems: a single ParaView file

Specialized

integrals over the domain, coefficients exported to the VTK

Putting it together: a Problem

1

Headers and
parallelism

2

Spatial
discretization

3

**Multiphysics
coupling**

4

Time
discretization

```
TransientPB ch_pb("CahnHilliard", // name
                 ch_oper,          // operator + integrators
                 ch_vars,          // primary variables  $\phi, \mu$ 
                 {ch_coef, ch_coef}, // one Coefficients per variable
                 ch_pst,          // post-processing (optional)
                 ac_vars);        // auxiliary  $\eta_1 \dots \eta_N$  (optional)
```

**As many Coefficients as
variables**

the vector has the size of VARS

In the same order

the i-th Coefficients goes with the
i-th variable

Stop at steady state

optionally, a Convergence object on
each variable

SteadyPB: same construction, with a SteadyOPE.

Chaining problems: a Coupling

1

Headers and
parallelism

2

Spatial
discretization

3

Multiphysics
coupling

4

Time
discretization

Coupling “CH/AC”

ac_pb
solved 1st



ch_pb
solved 2nd

Partitioned scheme (1 step): one problem after
another, in the order written.

Monolithic within a problem

Cahn-Hilliard ϕ and μ are solved together

```
// two different problems  
auto cc = Coupling("CH/AC", ac_pb, ch_pb);
```

```
// N identical problems  
std::vector<TransientPB> ac_pbs;  
// ... fill the N problems ...  
auto cc = setCoupling<N>("Multigrains", ac_pbs);
```

Caution: N must equal the size of the vector.

Resolution: time marching + run

1

Headers and
parallelism

2

Spatial
discretization

3

Multiphysics
coupling

4

Time
discretization

```
auto params = Parameters(  
    Parameter("initial_time", 0.),  
    Parameter("final_time", t_final),  
    Parameter("time_step", dt));  
auto time = TimeDiscretization(params, cc);  
time.solve();    // runs the simulation
```

Several couplings: `TimeDiscretization(params, cc1, cc2, ...)`, solved one after another.

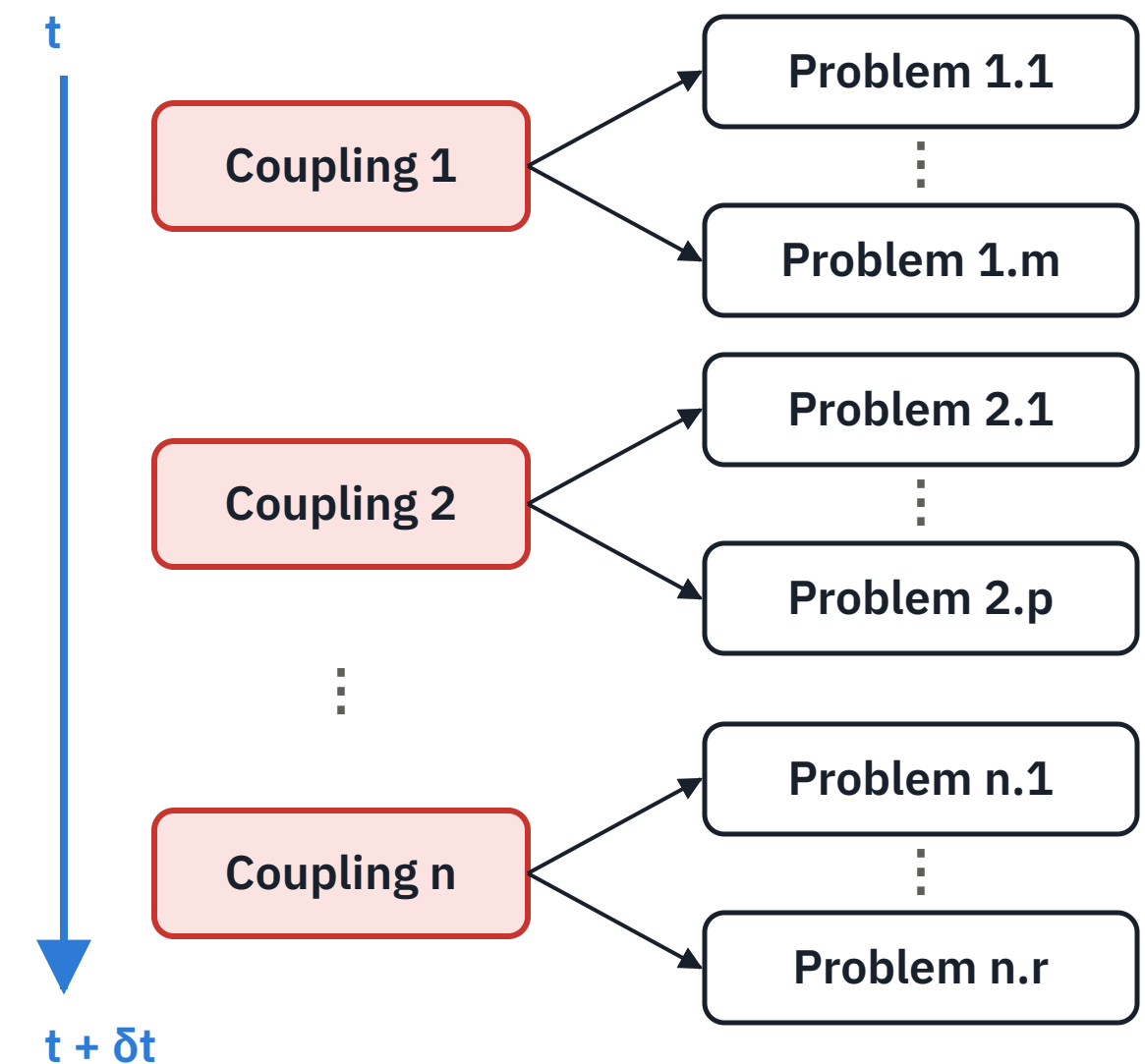
Variable time step (but not adaptive)

a `std::function<double(double)>` replaces `time_step`

Restart

`initial_iteration` restarts from a saved state

One time step



From the simple case to the multiphase case

Same skeleton: only part 3 changes

1 Headers
unchanged

2 Space
unchanged

3 Coupling
changes

4 Time
unchanged

One problem per equation

SIMPLE CASE

One Problem: Allen-Cahn.

MULTIPHASE CASE

Cahn-Hilliard (φ, μ) + Allen-Cahn ($\eta_1 \dots \eta_N$), in one Coupling.

Auxiliary variables

SIMPLE CASE

None.

MULTIPHASE CASE

Each problem reads the other's fields without solving them.

An energy you write

SIMPLE CASE

$W(\omega), H()$: provided by SLOTH.

MULTIPHASE CASE

$f(\varphi, \eta_1 \dots \eta_N)$ in the JSON, differentiated by Sympy.

Where to find information

A path in four steps



STEP 1

Get started

Getting started

install and build

Basic features

the structure of an input file



STEP 2

Get inspired

Examples

commented cases

tests/ in the repository

complete main.cpp files



STEP 3

Go deeper

User manual

all the options

PDEs

operators and integrators

Tutorials

AMR, save/restart



STEP 4

Get help

mfem.org

numerics and MPI

GitHub issues

bugs and questions

`collab4sloth.github.io/Documentation`