Technical university of Liberec

Faculty of mechatronics, informatics
and interdisciplinary studies

Flow123d

version 1.8.2_rc

**Documentation of file formats
and brief user manual.**

Liberec, 2013

Authors:

Jan Březina, Jan Stebel, Jiří Hnídek, David Flanderka, Pavel Exner, Lukáš Zedek

Acknowledgement

This work was supported by the Technology Agency of the Czech Republic under the project no. TA01021331.

Contents

Chapter 1

Quick start

Flow123D is a software for simulation of water flow, reactionary solute transport and heat transfer in a heterogeneous porous and fractured medium. In particular it is suited for simulation of underground processes in a granite rock massive. The program is able to describe explicitly processes in 3D medium, 2D fractures, and 1D channels and exchange between domains of different dimensions. The computational mesh is therefore a collection of tetrahedra, triangles and line segments.

The water flow model assumes a saturated medium described by the Darcy law. For discretization, we use lumped mixed-hybrid finite element method. We support both steady and unsteady water flow. The water flow model can be sequentially coupled with two different models for a solute transport or with a heat transfer model.

The first solute transport model can deal only with pure advection of several substances without any diffusion-dispersion term. It uses explicit Euler method for time discretization and finite volume method for space discretization and operator splitting method to couple with various processes described by the reaction term. The reaction term can treat any meaningful combination of the dual porosity, adsorptions, decays and linear reactions. Alternatively, one can use interface to the experimental SEMCHEM package for more complex geochemistry.

The second solute transport model describes general advection with hydrodynamic dispersion for several substances. It uses implicit Euler method for time discretization and discontinuous Galerkin method of the first, second or third order for the discretization in space. Currently there is no support for reaction term, the operator splitting approach (although it is not suited for implicit time schemes) is planned for the next version.

The heat transfer model assumes equilibrium between temperature of the rock and the fluid phase. It uses the same numerical scheme as the second transport model.

The program support output of all input and many output fields into two file formats. You can use file format of GMSH mesh generator and post-processor or you can use output into widely supported VTK format. In particular we recommend Paraview software for visualization and post-processing of the VTK data.

The program is implemented in C/C++ using essentially PETSC library for linear algebra. All models can run in parallel using MPI environment, however, the scalability of the whole program is limited due to serial mesh and serial outputs.

The program is distributed under GNU GPL v. 3 license and is available on the project

web page: http://flow123d.github.io

with sources on the GitHub: https://github.com/flow123d/flow123d.

1.1 Basic usage

1.1.1 How to run the simulation.

On the Linux system the program can be started either directly or through a script `flow123d.sh`, both placed in the `bin` directory of the installation package or of the source tree. When started directly, e.g. by the command

```
> flow123d -s example.con
```

the program requires one argument after switch `-s` which is the name of the principal input file. Full list of possible command line arguments is as follows.

`--help`
: Parameters interpreted by Flow123d. Remaining parameters are passed to PETSC.

`-s, --solve <file>`
: Set principal CON input file. All relative paths in the CON file are relative against current directory.

`-i, --input_dir <directory>`
: The placeholder `${INPUT}` used in the path of an input file will be replaced by the `<directory>`. Default value is `input`.

`-o, --output_dir <directory>`
: All paths for output files will be relative to this `<directory>`. Default value is `output`.

`-l, --log <file_name>`
: Set base name of log files. Default value is `flow123d`. The log files are individual for every MPI process, placed in the output directory. The MPI rank of the process and the `log` suffix are appended to the base name.

`--no_log`
: Turn off logging.

`--no_profiler`
: Turn off profiler output.

`--full_doc`
: Prints full structure of the main input file.

`--JSON_template`
: Prints a description of the main input file as a valid CON file template.

`--latex_doc`
: Prints a description of the main input file in LaTeX format using particular macros.

All other parameters will be passed to the PETSC library. An advanced user can influence lot of parameters of linear solvers. In order to get list of supported options use parameter `-help` together with some valid input. Options for various PETSC modules are displayed when the module is used for the first time.

Alternatively, you can use script `flow123d.sh` to start parallel jobs or limit resources used by the program. The syntax is as follows:

```
flow123d.sh [OPTIONS] -- [FLOW_PARAMS]
```

where everything after double dash is passed as parameters to the flow123d binary. The script accepts following options:

`-h, --help`
: Usage overview.

`--host <hostname>`
: Default value is the host name obtained by system `hostname` command, this argument can be used to override it. Resulting value is used to select a backend script `config/<hostname>.sh`, which describes particular method how to start parallel jobs, usually through some sort of PBS job queue system. If the script is not found, we try to start parallel processes directly on the actual host."

`-t, --walltime <timeout>`
: Upper estimate for real running time of the calculation. Kill calculation after *timeout* seconds. Can also be used by PBS to choose appropriate job queue.

`-np <number of processes>`
: Specify number of MPI parallel processes for calculation.

`-m, --mem <memory limit>`
: Limits total available memory to `<memory limit>` bytes per process.

`-n, --nice <niceness>`
: Change priority of the calculation, higher values means lower priority. See the `nice` command.

`-ppn <processes per node>`
: Set number of processes started on one node for multicore systems. Number of processes set by `-np` parameter should be divisible by `<processes per node>`.

`-q, --queue <queue>`
: Select particular job queue on PBS systems. If running without PBS, it redirects stdout and stderr to the file `<queue>.<date>`, which appended date and time of the start of the job.

On the windows operating systems, we use Cygwin libraries in order to emulate Linux API. Therefore you have to keep the Cygwin libraries within the same directory as the program executable. The Windows package that can be downloaded from project web page contains both the Cygwin libraries and the mpiexec command for starting parallel jobs on the windows based workstations.

Then you can start the sequential run by the command:

```
> flow123d.exe -s example.con
```

or the parallel run by the command:

```
> mpiexec.exe -np 2 flow123d.exe -s example.con
```

The program accepts the same parameters as the Linux version, but there is no script similar to `flow123d.sh` for the windows operating systems.

1.1.2 Tutorial problem

CON file format

The main input file uses a slightly extended JSON file format which together with some particular constructs forms a CON (C++ object notation) file format. Main extensions of the JSON are unquoted key names (as long as they do not contain whitespaces), possibility to use `=` instead of `:` and C++ comments, i.e. `//` for a one line and `/* */` for a multi-line comment. In CON file format, we prefer to call JSON objects "records" and we introduce also "abstract records" that mimic C++ abstract classes, arrays of a CON file have only elements of the same type (possibly using abstract record types for polymorphism). The usual keys are in lower case and without spaces (using underscores instead), there are few special upper case keys that are interpreted by the reader: `REF` key for references, `TYPE` key for specifing actual type of an abstract record. For detailed description see Section 3.1.

Geometry

In the following, we shall provide a commented input for the tutorial problem:

`tests/03_transport_small_12d/flow_vtk.con`

We consider a simple 2D problem with a branching 1D fracture (see Figure 1.1 for the geometry). To prepare a mesh file we use the GMSH software. First, we construct a geometry file. In our case the geometry consists of:

- one physical 2D domain corresponding to the whole square
- three 1D physical domains of the fracture
- four 1D boundary physical domains of the 2D domain
- three 0D boundary physical domains of the 1D domain

In this simple example, we can in fact combine physical domains in every group, however we use this more complex setting for demonstration purposes. Using GMSH graphical interface we can prepare the GEO file where physical domains are referenced by numbers, then we use any text editor and replace numbers with string labels in such a way that the labels of boundary physical domains start with the dot character. These are the domains where we will not do any calculations but we will use them for setting boundary conditions. Finally, we get the GEO file like this:

```
1  cl1 = 0.16;
2  Point(1) = {0, 1, 0, cl1};
3  Point(2) = {1, 1, 0, cl1};
4  Point(3) = {1, 0, 0, cl1};
5  Point(4) = {0, 0, 0, cl1};
6  Point(6) = {0.25, -0, 0, cl1};
7  Point(7) = {0, 0.25, 0, cl1};
8  Point(8) = {0.5, 0.5, -0, cl1};
9  Point(9) = {0.75, 1, 0, cl1};
10 Line(19) = {9, 8};
11 Line(20) = {7, 8};
12 Line(21) = {8, 6};
13 Line(22) = {2, 3};
14 Line(23) = {2, 9};
15 Line(24) = {9, 1};
16 Line(25) = {1, 7};
17 Line(26) = {7, 4};
18 Line(27) = {4, 6};
19 Line(28) = {6, 3};
20 Line Loop(30) = {20, -19, 24, 25};
21 Plane Surface(30) = {30};
22 Line Loop(32) = {23, 19, 21, 28, -22};
23 Plane Surface(32) = {32};
24 Line Loop(34) = {26, 27, -21, -20};
25 Plane Surface(34) = {34};
26 Physical Point(".1d_top") = {9};
27 Physical Point(".1d_left") = {7};
28 Physical Point(".1d_bottom") = {6};
29 Physical Line("1d_upper") = {19};
30 Physical Line("1d_lower") = {21};
31 Physical Line("1d_left_branch") = {20};
32 Physical Line(".2d_top") = {23, 24};
33 Physical Line(".2d_right") = {22};
34 Physical Line(".2d_bottom") = {27, 28};
35 Physical Line(".2d_left") = {25, 26};
36 Physical Surface("2d") = {30, 32, 34};
```

Notice the labeled physical domains on lines 26 – 36. Then we just set the discretization step `cl1` and use GMSH to create the mesh file. The mesh file contains both the 'bulk' elements where we perform calculations and the 'boundary' elements (on the boundary physical domains) where we only set the boundary conditions.

Having the computational mesh, we can create the main input file with the description of our problem.

```
{
  problem = {
    TYPE = "SequentialCoupling",
    description = "Transport 1D-2D, (convection, dual porosity, sorption)",
    mesh = {
      mesh_file = "./input/mesh_with_boundary.msh",
      sets = [
          { name="1d_domain",
            region_labels = [ "1d_upper", "1d_lower", "1d_left_branch" ]
          }
        ]
    },
```

The file starts with a selection of problem type (`SequentialCoupling`), and a textual problem description. Next, we specify the computational mesh, here it consists of the name of the mesh file and the declaration of one *region set* composed of all 1D regions i.e. representing the whole fracture. Other keys of the mesh record allow labeling regions given only by numbers, defining new regions in terms of element numbers (e.g to have leakage on single element), defining boundary regions, and set operations with region sets, see Section 3.2.1 for details.

Flow setting

Next, we setup the flow problem. We shall consider a flow driven only by the pressure gradient (no gravity), setting the Dirichlet boundary condition on the whole boundary with the pressure head equal to $x+y$. The conductivity will be $k_2 = 1$ on the 2D domain and $k_1 = 10$ on the 1D domain. The fracture width will be $\delta_1 = 1$ (default value) and the crosswind conductivity between dimensions will have value 1, which can be achieved by setting the dimensionless parameter $\sigma_2 = \frac{1}{2k_1} = 0.05$.

```
    primary_equation = {
      TYPE = "Steady_MH",

      input_fields = [
        { r_set = "1d_domain", conductivity = 10, sigma = 0.05 },
        { region = "2d",       conductivity = 1  },
        { r_set = "BOUNDARY",
          bc_type = "dirichlet",
          bc_pressure = { TYPE="FieldFormula", value = "x+y" }
        }
      ],

      output = {
        output_stream = { REF = "/system/output_streams/0" },
        output_fields = [ "pressure_p0", "pressure_p1", "velocity_p0" ]
```

```
28        },
29
30        solver = {
31          TYPE = "Petsc",
32          a_tol = 1e-12,
33          r_tol = 1e-12
34        }
35      }, // primary equation
```

On line 14, we specify particular implementation (numerical method) of the flow solver, in this case the Mixed-Hybrid solver for unsteady problems. On lines 16 – 23, we set mathematical fields that live on the computational domain as well as those for boundary conditions, we set only the conductivity field since other `input_fields` have appropriate default values. We use implicitly defined set "BOUNDARY" that contains all boundary regions and set there dirichlet boundary condition in terms of the pressure head. In this case, the field is not of the implicit type `FieldConstant`, so we must specify the type of the field `TYPE="FieldFormula"`. See Section 3.2.2 for other field types. On lines 25 – 28 we specify which output fields should be written to the output stream (that means particular output file, with given format). Currently, we support only one output stream per equation, so this allows at least switching individual output fields on or off. See Section 3.4 for the list of available `output_fields`. Notice the reference used on line 26 pointing to the definition of the output streams at the end of the file. Finally, we specify type of the linear solver and its tolerances.

Transport setting

We also consider subsequent transport problem with the porosity $\theta = 0.25$ and zero initial concentration. The boundary condition is equal to 1 and is automatically applied only on the inflow part of the boundary. There are also some adsorption and dual porosity models in this particular test case. Adsorption and simple reactions model inputs are particularly described in subsections 1.1.2.

```
36      secondary_equation = {
37        TYPE = "TransportOperatorSplitting",
38
39        substances = [ "age", "U235" ],
40
41        input_fields= [
42          { r_set = "ALL",
43            init_conc = 0,
44            porosity= 0.25
45          },
46          { r_set = "BOUNDARY",
47            bc_conc = 1.0
48          }
49        ],
50
51        reaction_term = {
```

```
52          TYPE = "DualPorosity",
53
54          input_fields= [
55            {
56              r_set="ALL",
57              diffusion_rate_immobile = [0.01,0.01],
58              porosity_immobile = 0.25,
59              init_conc_immobile = [0.0, 0.0]
60            }
61          ],
62
63          output_fields = [],
64
65          reaction_mobile = {
66            TYPE = "SorptionMobile",
67            solvent_density = 1.0,
68            substances = ["age", "U235"],
69            molar_mass = [1.0, 1.0],
70            solubility = [1.0, 1.0],
71
72            input_fields= [
73              {
74                r_set="ALL",
75                rock_density = 1.0,
76                sorption_type =  ["linear", "freundlich"],
77                isotherm_mult = 0.02,
78                isotherm_other = [0, 0.5]
79              }
80            ],
81            output_fields = []
82          },
83          reaction_immobile = {
84            TYPE = "SorptionImmobile",
85            solvent_density = 1.0,
86            substances = ["age", "U235"],
87            molar_mass = [1.0, 1.0],
88            solubility = [1.0, 1.0],
89            input_fields = { REF="../../reaction_mobile/input_fields" },
90            output_fields = []
91          }
92        },
93
94        output_stream = { REF = "/system/output_streams/1" },
95
96        time = { end_time = 1.0 },
97        mass_balance = { cumulative = true }
98      } // secondary_equation
99    }, // problem
```

For the transport problem we use implementation called "TransportOperatorSplitting" which stands for an explicit finite volume solver of the convection equation (without diffusion), the operator splitting is used for the equilibrium adsorption as well as for the dual porosity model and first order reactions simulation. On line 39, we set names of transported substances, here it is the age of the water and the uranium 235. On lines 41 – 49, we set the input fields, in particular the `porosity` and the initial concentrations (one for every substance). However, on line 43 we see only single value since an automatic conversion is applied to turn the scalar zero into the zero vector (of size 2). On lines 46 – 48, we set the boundary fields, namely the concentration on the inflow part of the boundary. We need not to specify the type of the condition since currently there is only one type for this transport model. On lines 51 – 92, we specify the dual porosity and adsorption mechanisms (see comments below). We also have to prescribe the time setting, here only the end time of the simulation since the step size is determined from the CFL condition; however a smaller time step can be enforced if necessary.

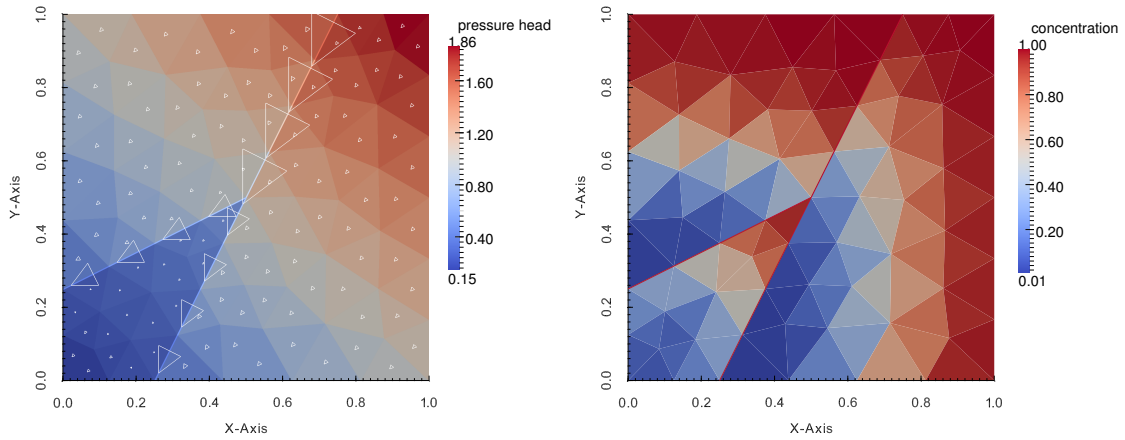
Sorption settings

The input information for dual porosity and equilibrial sorption are enclosed in the record `reaction_term`. The type of process is determined by `TYPE="DualPorosity"`. The `solvent_density` is supposed to be constant all over the simulated area. The vector `substances` contains the list of soluted substances whose concentrations are concidered to be affected by sorptions. Material characteristics of all the sorbing substances can be defined by vectors `molar_mass` and `solubility`. Elements of the vector `solubility` define the upper bound of an aqueous concentration which can appear, because some substances have limited solubility and if the solubility exceeds this limit they start to precipitate. `solubility` is a crucial parameter for solving further described set of nonlinear equations. The parameter `substeps` is important when interpolation is used to search approximative solution of the adsorption problem. It is the number of precomputed points lying on the isotherm. Its default value is 1000.

The record `input_fields` colects information about region specific parameters. Particular region (bulk Physical Entity), where one kind of adsorption takes place, can be specified by its label from gmsh-file. All implemented types of adsorption can take the rock density in region into account. The value of `rock_density` can be either constant or specified by `FieldFormula`. The `sorption_type` reperesents the empirically determined isotherm and can have one of four possible values: { `"none", "linear", "freundlich", "langmuir"` }. Linear isotherm needs just one parameter to be given whereas Freundlichs' and Langmuirs' isotherm have two parameters. For further details about mathematical description see Section 2.5.2. Isothermally described sorption simulation can be used in the case of low concentrated solutions without competition between multiple dissolved species.

Output streams and results

```
100   system = {
101     output_streams = [
102       {
103         file = "test3.pvd",
```



(a) Elementwise pressure head and velocity field (triangles).

(b) Propagation of U235 from the inflow part of the boundary.

Figure 1.1: Results of the tutorial problem.

```
104           format = { TYPE = "vtk", variant = "ascii" },
105           name = "flow_output_stream"
106         },
107         {
108           file = "test3-transport.pvd",
109           format = { TYPE = "vtk", variant = "ascii" },
110           time_step = 0.1,
111           name = "transport_output_stream"
112         }
113       ]
114     }
115   }
```

The end of the input file contains declaration of two output streams, one for the flow problem and one for the transport problem. Currently, we support output into VTK and GMSH data format. In the output record for time-dependent process we have to specify the `time_step` (line 110) which determines the frequency of saving. In Figure 1.1 one can see the results: the pressure and the velocity field on the left and the concentration of U235 at time $t = 0.9$ on the right. Even if the pressure gradient is the same in the 2D domain and in the fracture, due to higher conductivity the velocity field is ten times faster in the fracture. Since porosity is the same, the substance is transported faster by the fracture and then appears in the bottom left 2D domain before the main wave propagating solely through the 2D domain.

In the following chapter we describe mathematical models used in Flow123d. Then in chapter 3 we briefly describe structure of individual input files, in particular the main CON file. The complete description of the CON format is given in chapter 4.

Chapter 2

Mathematical models of physical reality

Flow123d provides models for Darcy flow in porous media as well as for the transport and reactions of solutes. In this section, we describe mathematical formulations of these models together with physical meaning and units of all involved quantities. In the first section we present basic notation and assumptions about computational domains and meshes that combine different dimensions. In the next section we derive approximation of thin fractures by lower dimensional interfaces for a general transport process. Latter sections describe details for models of particular physical processes.

2.1 Meshes of mixed dimension

Common and unique feature of all models in Flow123d is the support of domains with mixed dimension. Let $\Omega_3 \subset \mathbf{R}^3$ be an open set representing continuous approximation of porous and fractured medium. Similarly, we consider a 2D manifold $\Omega_2 \subset \overline{\Omega}_3$, representing 2D fractures and a 1D manifold $\Omega_1 \subset \overline{\Omega}_2$ of 1D channels or preferential paths (see Fig 2.1). We assume that Ω_2 and Ω_1 are polytopic (i.e. polygonal and piecewise linear, respectively). For every dimension $d = 1, 2, 3$, we introduce a triangulation $\mathcal{T}_d$ of the open set Ω_d that consists of finite elements T_d^i, $i = 1, \ldots, N_E^d$. The elements are simplices, i.e. lines, triangles and tetrahedra, respectively.

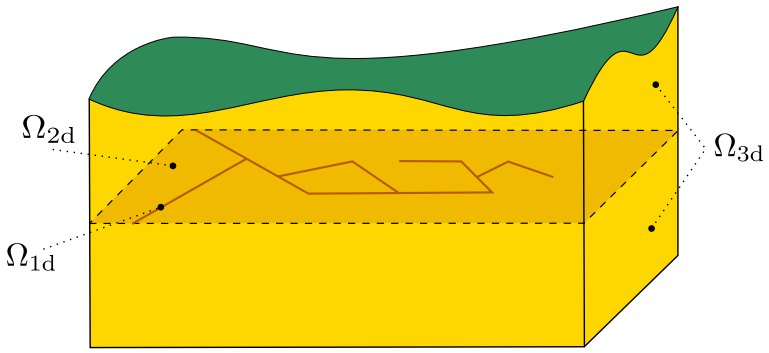


Figure 2.1: Scheme of a problem with domains of multiple dimensions.

Present numerical methods require meshes satisfying the compatibility conditions

$$T^i_{d-1} \cap T_d \subset \mathcal{F}_d, \qquad \text{where } \mathcal{F}_d = \bigcup_k \partial T^k_d \tag{2.1}$$

and

$$T^i_{d-1} \cap \mathcal{F}_d \text{ is either } T^i_{d-1} \text{ or } \emptyset \tag{2.2}$$

for every $i \in \{1, \dots, N_E^{d-1}\}$, $j \in \{1, \dots, N_E^d\}$, and $d = 2, 3$. That is, the $(d-1)$-dimensional elements are either between d-dimensional elements and match their sides or they poke out of Ω_d.

2.2 Advection-diffusion processes on fractures

In this section, we shall derive a model for a general advection-diffusion process in a domain of mixed dimensions using simple approximations inspired by the paper Martin et al. [4]. Let us consider a fracture as a strip domain

$$\Omega_f \subset [0, \delta] \times \mathbf{R}^{d-1}$$

for $d = 2$ or $d = 3$ and surrounding continuum domains

$$\Omega_1 \subset (-\infty, 0) \times \mathbf{R}^{d-1}, \Omega_2 \subset (\delta, \infty) \times \mathbf{R}^{d-1}.$$

Further, we denote by γ_i, $i = 1, 2$ the fracture faces common with domains Ω_1 and Ω_2 respectively. By x, $\boldsymbol{y}$ we denote normal and tangential coordinate of a point in Ω_f. We consider the normal vector $\boldsymbol{n} = \boldsymbol{n}_1 = -\boldsymbol{n}_2 = (1, 0, 0)^\top$. An advection-diffusion process is given by equations:

$$\begin{aligned}
\partial_t w_i + \operatorname{div} \boldsymbol{j}_i &= f_i && \text{on } \Omega_i,\ i = 1, 2, f, && (2.3)\\
\boldsymbol{j}_i &= -\mathbb{A}_i \nabla u_i + \boldsymbol{b}_i w_i && \text{on } \Omega_i,\ i = 1, 2, f, && (2.4)\\
u_i &= u_f && \text{on } \gamma_i,\ i = 1, 2, && (2.5)\\
\boldsymbol{j}_i \cdot \boldsymbol{n} &= \boldsymbol{j}_f \cdot \boldsymbol{n} && \text{on } \gamma_i,\ i = 1, 2, && (2.6)
\end{aligned}$$

where $w_i = w_i(u_i)$ is the conservative quantity and u_i is the principal unknown, $\boldsymbol{j}_i$ is the flux of w_i, f_i is the source term, $\mathbb{A}_i$ is the diffusivity tensor and $\boldsymbol{b}_i$ is the velocity field. We assume that the tensor $\mathbb{A}_f$ is symmetric positive definite with one eigenvector in the direction $\boldsymbol{n}$. Consequently the tensor has the form:

$$A_f = \begin{pmatrix} a_n & 0 \\ 0 & \mathbb{A}_t \end{pmatrix}$$

Furthermore, we assume that $\mathbb{A}_f(x, \boldsymbol{y}) = \mathbb{A}_f(\boldsymbol{y})$ is constant in the normal direction.

Our next aim is to integrate equations on the fracture Ω_f in the normal direction and obtain their approximations on the surface $\gamma = \Omega_f \cap \{x = \delta/2\}$ running through the middle of the fracture. For the sake of clarity, we will not write subscript f for quantities on the fracture. To make the following procedure mathematicaly correct we have to assume that functions $\partial_x w$, $\partial_x \nabla_{\boldsymbol{y}} u$, $\partial_x \boldsymbol{b}_{\boldsymbol{y}}$ are continuous and bounded on Ω_f. Here and later on $\boldsymbol{b}_x = (\boldsymbol{b} \cdot \boldsymbol{n})\, \boldsymbol{n}$ is the normal part of the velocity field and $\boldsymbol{b}_{\boldsymbol{y}} = \boldsymbol{b} - \boldsymbol{b}_x$ is the

tangential part. The same notation will be used for normal and tangential part of the field $\boldsymbol{q}$.

We integrate (2.3) over the fracture opening $[0, \delta]$ and use approximations to get

$$\partial_t(\delta W) - \boldsymbol{j}_2 \cdot \boldsymbol{n}_2 - \boldsymbol{j}_1 \cdot \boldsymbol{n}_1 + \operatorname{div} \boldsymbol{J} = \delta F, \tag{2.7}$$

where for the first term, we have used mean value theorem, first order Taylor expansion, and boundedness of $\partial_x w$ to obtain approximation:

$$\int_0^\delta w(x, \boldsymbol{y}) \, \mathrm{d}x = \delta w(\xi_{\boldsymbol{y}}, \boldsymbol{y}) = \delta W(\boldsymbol{y}) + O(\delta^2 |\partial_x w|),$$

where

$$W(\boldsymbol{y}) = w(\delta/2, \boldsymbol{y}) = w(u(\delta/2, \boldsymbol{y})) = w(U(\boldsymbol{y})).$$

Next two terms in (2.7) come from the exact integration of the divergence of the normal flux $\boldsymbol{j}_x$. Integration of the divergence of the tangential flux $\boldsymbol{j}_{\boldsymbol{y}}$ gives the fourth term, where we introduced

$$\boldsymbol{J}(\boldsymbol{y}) = \int_0^\delta \boldsymbol{j}_{\boldsymbol{y}}(x, \boldsymbol{y}) \, \mathrm{d}x.$$

In fact, this flux on γ is scalar for the case $d = 2$. Finally, we integrate the right-hand side to get

$$\int_0^\delta f(x, \boldsymbol{y}) \, \mathrm{d}x = \delta F(\boldsymbol{y}) + O(\delta^2 |\partial_x f|), \quad F(\boldsymbol{y}) = f(\delta/2, \boldsymbol{y}).$$

Due to the particular form of the tensor $\mathbb{A}_f$, we can separately integrate tangential and normal part of the flux given by (2.4). Integrating the tangential part and using approximations

$$\int_0^\delta \nabla_{\boldsymbol{y}} u(x, \boldsymbol{y}) \, \mathrm{d}x = \delta \nabla_{\boldsymbol{y}} u(\xi_{\boldsymbol{y}}, \boldsymbol{y}) = \delta \nabla_{\boldsymbol{y}} U(\boldsymbol{y}) + O\big(\delta^2 |\partial_x \nabla_{\boldsymbol{y}} u|\big)$$

and

$$\int_0^\delta \big(\boldsymbol{b}_{\boldsymbol{y}} w\big)(x, \boldsymbol{y}) \, \mathrm{d}x = \delta \boldsymbol{B}(\boldsymbol{y}) W(\boldsymbol{y}) + O\big(\delta^2 |\partial_x (\boldsymbol{b}_{\boldsymbol{y}} w)|\big)$$

where

$$\boldsymbol{B}(\boldsymbol{y}) = \boldsymbol{b}_{\boldsymbol{y}}(\delta/2, \boldsymbol{y}),$$

we obtain

$$\boldsymbol{J} = -\mathbb{A}_t \delta \nabla_{\boldsymbol{y}} U + \delta \boldsymbol{B} W + O\big(\delta^2 (|\partial_x \nabla_{\boldsymbol{y}} u| + |\partial_x (\boldsymbol{b}_{\boldsymbol{y}} w)|)\big). \tag{2.8}$$

So far, we have derived equations for the state quantities U and $\boldsymbol{J}$ on the fracture manifold γ. In order to get a well possed problem, we have to prescribe two conditions for boundaries γ_i, $i = 1, 2$. To this end, we perform integration of the normal flux $\boldsymbol{j}_x$, given by (2.4), separately for the left and right half of the fracture. Similarly as before we use approximations

$$\int_0^{\delta/2} \boldsymbol{j}_x \, \mathrm{d}x = (\boldsymbol{j}_1 \cdot \boldsymbol{n}_1) \frac{\delta}{2} + O(\delta^2 |\partial_x \boldsymbol{j}_x|)$$

and

$$\int_0^{\delta/2} \boldsymbol{b}_x w \,\mathrm{d}x = (\boldsymbol{b}_1 \cdot \boldsymbol{n}_1)\tilde{w}_1 \frac{\delta}{2} + O(\delta^2|\partial_x \boldsymbol{b}_x||w| + \delta^2|\boldsymbol{b}_x||\partial_x w|)$$

and their counter parts on the interval $(\delta/2, \delta)$ to get

$$\boldsymbol{j}_1 \cdot \boldsymbol{n}_1 = -\frac{2a_n}{\delta}(U - u_1) + \boldsymbol{b}_1 \cdot \boldsymbol{n}_1 \tilde{w}_1 \tag{2.9}$$

$$\boldsymbol{j}_2 \cdot \boldsymbol{n}_2 = -\frac{2a_n}{\delta}(U - u_2) + \boldsymbol{b}_2 \cdot \boldsymbol{n}_2 \tilde{w}_2 \tag{2.10}$$

where $\tilde{w}_i$ can be any convex combination of w_i and W. Equations (2.9) and (2.10) have meaning of a semi-discretized flux from domains Ω_i into fracture. In order to get a stable numerical scheme, we introduce a kind of upwind already on this level using a different convex combination for each flow direction:

$$\begin{aligned} \boldsymbol{j}_i \cdot \boldsymbol{n}_i = &- \sigma_i(U - u_i) \\ &+ \big[\boldsymbol{b}_i \cdot \boldsymbol{n}_i\big]^+ \big(\xi w_i + (1-\xi)W\big) \\ &+ \big[\boldsymbol{b}_i \cdot \boldsymbol{n}_i\big]^- \big((1-\xi)w_i + \xi W\big), \qquad i = 1, 2 \end{aligned} \tag{2.11}$$

where $\sigma_i = \frac{2a_n}{\delta}$ is the transition coefficient and the parameter $\xi \in [\frac{1}{2}, 1]$ can be used to interpolate between upwind ($\xi = 1$) and central difference ($\xi = \frac{1}{2}$) scheme. Equations (2.7), (2.8), and (2.11) describe the general form of the advection-diffusion process on the fracture and its communication with the surrounding continuum which we shall later apply to individual processes.

2.3 Darcy flow model

We consider simplest model for the velocity of the steady or unsteady flow in porous and fractured medium given by Darcy flow:

$$\boldsymbol{w} = -\mathbb{K}\nabla H \quad \text{on } \Omega_d, \text{ for } d = 1, 2, 3. \tag{2.12}$$

We drop the dimension index of quantities in equations if it is the same as the dimension of the set where the equation holds. In (2.12), $\boldsymbol{w}_d$ [ms^{-1}] is the superficial velocity, $\mathbb{K}_d$ is the conductivity tensor, and H_d [m] is the piezometric head. The velocity is related to the flux $\boldsymbol{q}_d$ with units [m^{4-d}s^{-1}] through

$$\boldsymbol{q}_d = \delta_d \boldsymbol{w}_d.$$

where δ_d [m^{3-d}] is a cross section coefficient, in particular $\delta_3 = 1$, δ_2 [m] is the thickness of a fracture, and δ_1 [m^2] is the cross-section of a channel. The flux q_d is the volume of the liquid (water) that pass through a unit square ($d = 3$), unit line ($d = 2$), or through a point ($d = 1$) per one second. The conductivity tensor is given by the product $\mathbb{K}_d = k_d \mathbb{A}_d$, where $k_d > 0$ is the hydraulic conductivity [ms^{-1}] and $\mathbb{A}_d$ is 3×3 dimensionless anisotropy tensor which has to be symmetric and positive definite. The piezometric-head H_d is related to the pressure head h_d by $H_d = h_d + z$ assuming that the gravity force acts in negative direction of the z-axes. Combining these relations we get the Darcy law in the form:

$$\boldsymbol{q} = -\delta k \mathbb{A} \nabla(h + z) \qquad \text{on } \Omega_d, \text{ for } d = 1, 2, 3. \tag{2.13}$$

Next, we employ continuity equation for saturated porous medium and the dimensional reduction from the preceding section (with $w = u := H$, $\boldsymbol{j} := \boldsymbol{w}$, $\mathbb{A} := \mathbb{K}$ and $\boldsymbol{b} := \boldsymbol{0}$), which yields:

$$\partial_t(\delta S\, h) + \operatorname{div}\boldsymbol{q} = F \qquad \text{on } \Omega_d, \text{ for } d = 1, 2, 3, \tag{2.14}$$

where S_d [m^{-1}] is the storativity and F_d [$\mathrm{m}^{3-d}\mathrm{s}^{-1}$] is a source term. In our setting the principal unknowns of the system (2.13, 2.14) are the pressure head h_d and the flux $\boldsymbol{q}_d$.

The storativity $S_d > 0$ or the volumetric specific storage can be expressed as

$$S_d = \gamma_w(\beta_r + \nu\beta_w), \tag{2.15}$$

where γ_w [$\mathrm{kgm}^{-2}\mathrm{s}^{-2}$] is the specific weight of water, ν is the porosity [$-$], β_r is compressibility of the bulk material of the pores (rock) and β_w is compressibility of the water both with units [$\mathrm{kg}^{-1}\mathrm{ms}^{-2}$]. For steady problems we set $S_d = 0$ for all dimensions $d = 1, 2, 3$. The source term F_d on the right hand side of (2.14) consists of the volume density of prescribed sources f_d [s^{-1}] and flux from higher dimension. Exact formula is slightly different for every dimension and will be discussed presently.

On Ω_3 we simply have $F_3 = f_3$ [s^{-1}].

On the set $\Omega_2 \cap \Omega_3$ the fracture is surrounded by one 3D surface from every side (or just one surface since we allow also 2D models on the boundary). On $\partial\Omega_3 \cap \Omega_2$ we prescribe boundary condition of Robin type

$$\begin{aligned} \boldsymbol{q}_3 \cdot \boldsymbol{n}^+ &= q_{32}^+ = \sigma_3(h_3^+ - h_2), \\ \boldsymbol{q}_3 \cdot \boldsymbol{n}^- &= q_{32}^- = \sigma_3(h_3^- - h_2), \end{aligned}$$

where $\boldsymbol{q}_3 \cdot \boldsymbol{n}^{+/-}$ [ms^{-1}] is the outflow from Ω_3, $h_3^{+/-}$ is a trace of the pressure head on Ω_3, h_2 is the pressure head on Ω_2, and σ_3 [s^{-1}] is the transition coefficient given by (see section 2.2 and [4])

$$\sigma_3 = \sigma_{32}\frac{2\mathbb{K}_2 : \boldsymbol{n}_2 \otimes \boldsymbol{n}_2}{\delta_2}.$$

Here $\boldsymbol{n}_2$ is the unit normal to the fracture (sign doesn't matter). On the other hand, the sum of the interchange fluxes $q_{32}^{+/-}$ forms a volume source on Ω_2. Therefore F_2 [ms^{-1}] on the right hand side of (2.14) is given by

$$F_2 = \delta_2 f_2 + (q_{32}^+ + q_{32}^-). \tag{2.16}$$

The communication between Ω_2 and Ω_1 is similar. However, in the 3D ambient space, an 1D channel can join multiple 2D fractures $1, \ldots, n$. Therefore, we have n independent outflows from Ω_2:

$$\boldsymbol{q}_2 \cdot \boldsymbol{n}^i = q_{21}^i = \sigma_2(h_2^i - h_1),$$

where σ_2 [ms^{-1}] is the transition coefficient integrated over the width of the fracture i:

$$\sigma_2 = \sigma_{21}\frac{2\delta_2^2\mathbb{K}_1 : \boldsymbol{n}_1^i \otimes \boldsymbol{n}_1^i}{\delta_1}.$$

Here $\boldsymbol{n}_1^i$ is the unit normal to the channel that is tangential to the fracture i. Sum of the fluxes forms part of F_1 [$\mathrm{m}^2\mathrm{s}^{-1}$]

$$F_1 = \delta_1 f_1 + \sum_{i=1}^{n} q_{21}^i. \tag{2.17}$$

We remark that the direct communication between 3D and 1D is neglected. The transition coefficients σ_{32} [−] and σ_{21} [−] are independent scaling parameters which represent the ratio of the crosswind and the tangential conductivity in the fracture.

In order to obtain unique solution we have to prescribe boundary conditions. Currently we support three basic types of boundary condition. Consider disjoint decomposition of the boundary

$$\partial\Omega_d = \Gamma_d^D \cap \Gamma_d^N \cap \Gamma_d^R$$

into Dirichlet, Neumann, and Robin parts. We prescribe

$$h_d = h_d^D \qquad \text{on } \Gamma_d^D, \tag{2.18}$$

$$\boldsymbol{q}_d \cdot \boldsymbol{n} = q_d^N \qquad \text{on } \Gamma_d^N, \tag{2.19}$$

$$\boldsymbol{q}_d \cdot \boldsymbol{n} = \sigma_d^R(h_d - h_d^R) \qquad \text{on } \Gamma_d^R. \tag{2.20}$$

where h_d^D, h_d^R is the given pressure head [m], which alternatively can be prescribed through the piezometric head H_d^D, H_d^R respectively. q_d^N is the given surface density of the boundary outflow [$\mathrm{m}^{4-d}\mathrm{s}^{-1}$], and σ_d^R is the transition coefficient [$\mathrm{m}^{3-d}\mathrm{s}^{-1}$]. The problem is well posed only if there is Dirichlet or Robin boundary condition on every component of the set $\Omega_1 \cup \Omega_2 \cup \Omega_3$ and $\sigma_d > 0$ for $d = 2, 3$.

For unsteady problems one has to specify initial condition in terms of initial pressure head h_d^0 or initial piezometric head H_d^0.

2.4 Transport of substances

The motion of substances dissolved in water is governed by the *advection*, and the *hydrodynamic dispersion*. In Ω_d, $d \in \{1, 2, 3\}$, we consider the following system of mass balance equations[1]:

$$\partial_t(\delta\vartheta c^i) + \operatorname{div}(\boldsymbol{q}c^i) - \operatorname{div}(\vartheta\delta\mathbb{D}^i\nabla c^i) = F_S^i + F_C^c(c^i) + F_R(c^1, \ldots, c^s). \tag{2.21}$$

The principal unknown is the concentration c^i [kgm^{-3}] of a substance $i \in \{1, \ldots, s\}$, which means weight of the substance in unit volume of the water. Other quantities are:

- ϑ [−] is the porosity, i.e. fraction of space occupied by water and the total volume.
- The hydrodynamic dispersivity tensor $\mathbb{D}^i$ [$\mathrm{m}^2\mathrm{s}^{-1}$] has the form

$$\mathbb{D}^i = D_m^i\tau\mathbb{I} + |\boldsymbol{v}|\left(\alpha_T^i\mathbb{I} + (\alpha_L^i - \alpha_T^i)\frac{\boldsymbol{v}\otimes\boldsymbol{v}}{|\boldsymbol{v}|^2}\right), \tag{2.22}$$

  which represents (isotropic) molecular diffusion, and mechanical dispersion in longitudal and transversal direction to the flow. Here D_m^i [$\mathrm{m}^2\mathrm{s}^{-1}$] is the molecular diffusion coefficient of the i-th substance (usual magnitude in clear water is 10^{-9}), $\tau = \vartheta^{1/3}$ is the tortuosity (by [5]), α_L^i [m] and α_T^i [m] is the longitudal dispersivity and the transverse dispersivity, respectively. Note that although we allow dispersivities to have different values for different substances, it is often assumed

[1] For $d \in \{1, 2\}$ this form can be derived as in Section 2.2 using $w := \delta\vartheta c^i$, $u := c^i$, $\mathbb{A} := \delta\vartheta\mathbb{D}^i$, $\boldsymbol{b} := \boldsymbol{v} = \frac{\boldsymbol{q}}{\vartheta\delta}$.

that they are intrinsic parameters of the porous medium. Finally, $\boldsymbol{v}$ [ms^{-1}] is the *microscopic* water velocity, related to the Darcy flux $\boldsymbol{q}$ by the relation $\boldsymbol{q} = \vartheta\delta\boldsymbol{v}$. The value of D_m^i for specific substances can be found in literature (see e.g. [1]). For instructions on how to determine α_L^i, α_T^i we refer to [2, 3].

- F_S^i [kgm^{-d}s^{-1}] represents the density of concentration sources. Its form is:

$$F_S^i = \delta f_S^i + \delta(c_S^i - c^i)\sigma_S. \tag{2.23}$$

  Here f_S^i [kgm^{-3}s^{-1}] is the density of concentration sources, c_S^i [kgm^{-3}] is an equilibrium concentration and σ_S^i [s^{-1}] is the concentration flux.

- $F_C^c(c^i)$ [kgm^{-d}s^{-1}] is the density of concentration sources due to exchange between regions with different dimensions, see (2.25) below.

- The reaction term $F_R(\dots)$ [kgm^{-d}s^{-1}] is thoroughly described in the next section 2.5.

Initial and boundary conditions. At time $t = 0$ the concentration is determined by the initial condition

$$c^i(0, \boldsymbol{x}) = c_0^i(\boldsymbol{x}).$$

The physical boundary $\partial\Omega_d$ is decomposed into the parts $\Gamma_I \cup \Gamma_D \cup \Gamma_N \cup \Gamma_R$, which may change during simulation time. The first part Γ_I is further divided into two segments:

$$\Gamma_I^+(t) = \{\boldsymbol{x} \in \partial\Omega_d \,|\, \boldsymbol{q}(t, \boldsymbol{x}) \cdot \boldsymbol{n}(\boldsymbol{x}) < 0\},$$
$$\Gamma_I^-(t) = \{\boldsymbol{x} \in \partial\Omega_d \,|\, \boldsymbol{q}(t, \boldsymbol{x}) \cdot \boldsymbol{n}(\boldsymbol{x}) \geq 0\},$$

where $\boldsymbol{n}$ stands for the unit outward normal vector to $\partial\Omega_d$. We prescribe the following boundary conditions: On the inflow parts $\Gamma_I^+ \cup \Gamma_D$, the user must provide Dirichlet boundary condition c_D^i for concentrations:

$$c^i = c_D^i \text{ on } \Gamma_I^+ \cup \Gamma_D.$$

On Γ_I^- we impose homogeneous Neumann boundary condition:

$$-\vartheta\delta\mathbb{D}^i\nabla c^i \cdot \boldsymbol{n} = 0 \text{ on } \Gamma_I^-,$$

on Γ_N we impose Neumann boundary condition with user-defined concentration flux f_N^i:

$$-\vartheta\delta\mathbb{D}^i\nabla c^i \cdot \boldsymbol{n} = f_N^i \text{ on } \Gamma_N,$$

and finally on Γ_R we impose Robin boundary condition through transition parameter σ_R^i and reference concentration c_D^i:

$$-\vartheta\delta\mathbb{D}^i\nabla c^i \cdot \boldsymbol{n} = \sigma_R^i(c^i - c_D^i) \text{ on } \Gamma_R.$$

Communication between dimensions. Transport of substances is considered also on interfaces of physical domains with adjacent dimensions (i.e. 3D-2D and 2D-1D, but not 3D-1D). Denoting c_{d+1}, c_d the concentration of a given substance in Ω_{d+1} and Ω_d, respectively, the comunication on the interface between Ω_{d+1} and Ω_d is described by the quantity

$$q^c_{d+1,d} = \sigma^c_{d+1,d} \frac{\delta^2_{d+1}}{\delta_d} 2\vartheta_d \mathbb{D}_d : \boldsymbol{n} \otimes \boldsymbol{n}(c_{d+1} - c_d) + \begin{cases} q^l_{d+1,d} c_{d+1} & \text{if } q^l_{d+1,d} \geq 0, \\ q^l_{d+1,d} \frac{\vartheta_d}{\vartheta_{d+1}} c_d & \text{if } q^l_{d+1,d} < 0, \end{cases} \tag{2.24}$$

where

- $q^c_{d+1,d}$ [kgm^{-d}s^{-1}] is the density of concentration flux from Ω_{d+1} to Ω_d,
- $\sigma^c_{d+1,d}$ [$-$] is a transition parameter. Its value determines the mass exchange between dimensions whenever the concentrations differ. In general, it is recommended to leave the default value $\sigma^c = 1$ or to set $\sigma^c = 0$ (when exchange is due to water flux only).
- $q^l_{d+1,d}$ [m^{3-d}s^{-1}] is the water flux from Ω_{d+1} to Ω_d, i.e. $q^l_{d+1,d} = \boldsymbol{q}_{d+1} \cdot \boldsymbol{n}_{d+1}$.

The communication between dimensions is incorporated as the total flux boundary condition for the problem on Ω_{d+1}:

$$-\vartheta\delta\mathbb{D}\nabla c \cdot \boldsymbol{n} + q^w c = q^c \tag{2.25}$$

and a source term in Ω_d:

$$F^c_{C3} = 0, \quad F^c_{C2} = q^c_{32}, \quad F^c_{C1} = q^c_{21}. \tag{2.26}$$

Mass balance. The advection-dispersion equation satisfies the balance of mass in the following form:

$$m^i(0) + \int_0^t s^i(\tau)\, d\tau - \int_0^t f^i(\tau)\, d\tau = m^i(t)$$

for any instant t in the computational time interval and any substance i. Here

$$m^i(t) := \sum_{d=1}^{3} \int_{\Omega^d} (\delta\vartheta c^i)(t, \boldsymbol{x})\, d\boldsymbol{x},$$

$$s^i(t) := \sum_{d=1}^{3} \int_{\Omega^d} F^i_S(t, \boldsymbol{x})\, d\boldsymbol{x},$$

$$f^i(t) := \sum_{d=1}^{3} \int_{\partial\Omega^d} \left(\boldsymbol{q}c^i - \vartheta\delta\mathbb{D}^i\nabla c^i\right)(t, \boldsymbol{x}) \cdot \boldsymbol{n}\, d\boldsymbol{x}$$

is the mass [kg], the volume source [kgs^{-1}] and the mass flux [kgs^{-1}] of i-th substance at time t, respectively. The mass, flux and source on every geometrical region is calculated at each computational time step and the values together with the control sums are written to the file `mass_balance.txt`.

Two transport models. Within the above presented model, Flow123d presents two possible approaches to solute transport.

- For modelling pure advection ($\mathbb{D} = 0$) one can choose `TransportOperatorSplitting` method, which represents an explicit in time finite volume solver. The solution process for one time step is faster, but the maximal time step is restricted. The resulting concentration is piecewise constant on mesh elements. This solver supports reaction term (involving simple chemical reactions, dual porosity and adsorption).
- The full model including dispersion is solved by `SoluteTransport_DG`, an implicit in time discontinuous Galerkin solver. It has no restriction of the computational time step and the space approximation is piecewise polynomial, currently up to order 3. Reaction term is currently not implemented.

2.5 Reaction term in transport

The `TransportOperatorSplitting` method supports the reaction term $F_R(c^1, \dots, c^s)$ on the right hand side of the equation (2.21). It can represent several models of chemical or physical nature. Figure 2.2 shows all possible reactional models that we support in combination with the transport process. The Operator Splitting method enables us to deal with the convection part and reaction term side by side. The convected quantities do not influence each other in the convectional process and are balanced over the elements. On the other hand the reaction term relates the convected quantities and can be computed separately on each element.

We move now to the description of the reaction models which can be seen again in Figure 2.2. The convected quantity is considered to be the concentration of substances. Up to now we can have *dual porosity*, *adsorption* (these two are more of a physical nature) and (chemical) *reaction* models in the reaction term.

The *reaction* model acts only on the specified substances and computes exchange of concentration among them. It does not have its own output because it only changes the concentration of substances in the specified zone where the reaction takes place. See **??** for thorough description.

The *adsorption* model describes the exchange of concentration of the substances between liquid and solid. It can be followed by another *reaction* that can run in both phases. The concentration in solid is an additional output of this model. See Subsection 2.5.2.

The *dual porosity* model, described in Subsection 2.5.1, introduces the so called immobile (or dead-end) pores in the matrix. The convection process operates only on the concentration of the substances in the mobile zone (open pores) and the exchange of concentrations from/to immobile zone is governed by molecular diffusion. This process can be followed by *adsorption* model and/or chemical *reaction*, both in mobile and immobile zone. The immobile concentration is an additional output.

2.5.1 Dual porosity

Up to now we have described the transport equation for the single porosity model. The dual porosity model splits the mass into two zones – the mobile zone and the immobile

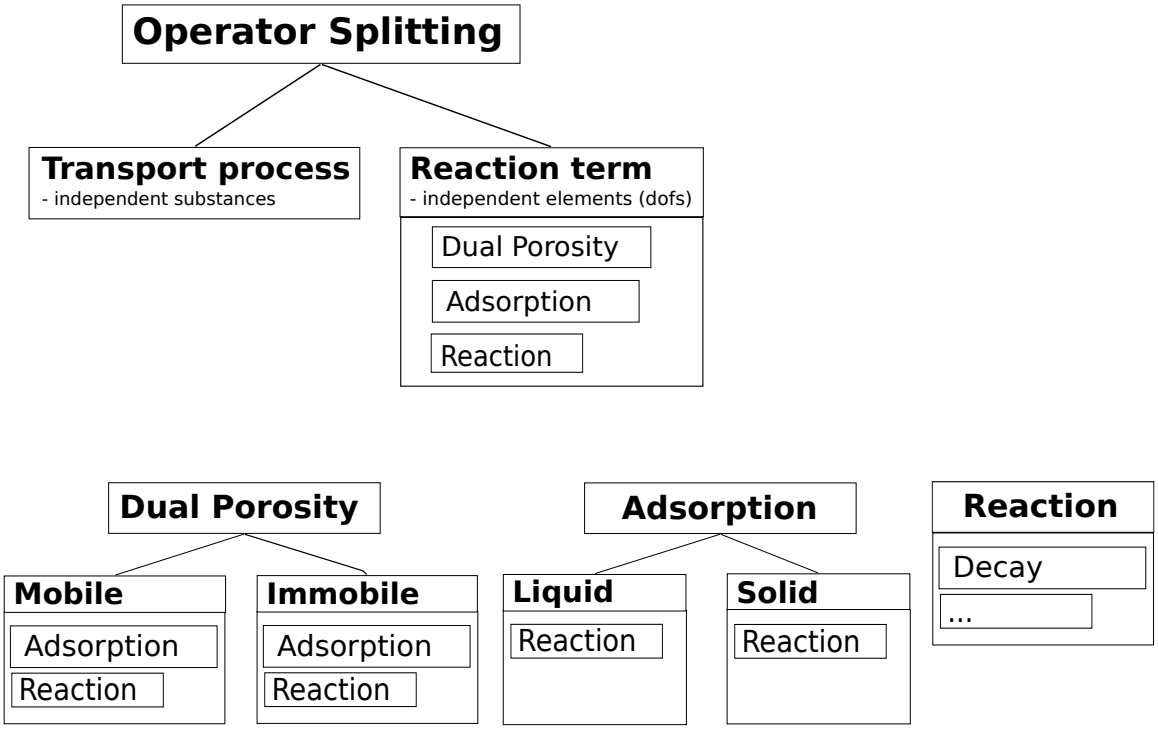


Figure 2.2: The scheme of the reaction term objects. The lines represents connections between different models. The tables under model name include the possible models which can be connected to the model above.

zone. Both occupy the same macroscopic volume, however on the microscopic scale, the immobile zone is formed by the dead-end pores, where the liquid is trapped and cannot pass through. The rest of the pore volume is occupied by the mobile zone. Since the liquid in the immobile pores is immobile, the exchange of the substance is only due to molecular diffusion. We consider simple nonequilibrium linear model:

$$\vartheta_m \partial_t c_m = D_{dp}(c_i - c_m), \tag{2.27}$$
$$\vartheta_i \partial_t c_i = D_{dp}(c_m - c_i), \tag{2.28}$$

where c_m is the concentration in the mobile zone, c_i is the concentration in the immobile zone and D_{dp} is a diffusion rate between the zones. ϑ_i denotes porosity of the immobile zone and $\vartheta_m = \vartheta$ the mobile porosity from transport equation (2.21).

The analytic solution of the system of differential equations at time t with initial conditions $c_m(0)$ and $c_i(0)$ is

$$c_m(t) = (c_m(0) - c_a(0)) \exp\left(-D_{dp}\left(\frac{1}{\vartheta_m} + \frac{1}{\vartheta_i}\right) t\right) + c_a(0), \tag{2.29}$$
$$c_i(t) = (c_i(0) - c_a(0)) \exp\left(-D_{dp}\left(\frac{1}{\vartheta_m} + \frac{1}{\vartheta_i}\right) t\right) + c_a(0), \tag{2.30}$$

where c_a is the weighted average

$$c_a = \frac{\vartheta_m c_m + \vartheta_i c_i}{\vartheta_m + \vartheta_i}.$$

If the time step is large we use the analytic solution to compute new values of concentrations. Otherwise we replace the time derivatives in (2.27) and (2.28) by first order forward differences and we get the classical Euler scheme

$$c_m(t^+) = \frac{D_{dp}\Delta t}{\vartheta_m}(c_i(t) - c_m(t)) + c_m(t), \tag{2.31}$$
$$c_i(t^+) = \frac{D_{dp}\Delta t}{\vartheta_i}(c_m(t) - c_i(t)) + c_i(t), \tag{2.32}$$
$$\tag{2.33}$$

where $\Delta t = t^+ - t$ is the time step.

The condition on the size of the time step is derived from the Taylor exapansion of (2.29) or (2.30), respectively. We neglect the higher order terms and we want the second order term smaller than given scheme tolerance *tol* relatively to c_a

$$(c_m(0) - c_a(0)) \frac{D_{dp}^2 (\Delta t)^2 \left(\frac{\vartheta_m + \vartheta_i}{\vartheta_m \vartheta_i}\right)^2}{2} \frac{1}{c_a} \leq tol. \tag{2.34}$$

Then we transform the above inequation into the following condition which is tested in the program

$$\max(|c_m(0) - c_a(0)|, |c_i(0) - c_a(0)|) \leq 2c_a \left(\frac{\vartheta_m \vartheta_i}{D_{dp}\Delta t(\vartheta_m + \vartheta_i)}\right)^2 tol. \tag{2.35}$$

If the inequation (2.35) is not satisfied then the analytic solution is used.

2.5.2 Equilibrial adsorption

The simulation of monolayer, equilibrial adsorption is based on the solution of two algebraic equations, namely the mass balance (in unit volume)

$$\vartheta \varrho_l c_l + (1-\vartheta)\varrho_s M_s c_s = c_T = const. \tag{2.36}$$

and an empirical adsorption law

$$c_s = f(c_l), \tag{2.37}$$

given in terms of the so-called isotherm f. Its form is determined by the parameter `sorption_type`:

- "*none*": $f(c_l) = 0$ (the adsorption model returns zero concentration in solid);
- "*linear*": $f(c_l) = k_l c_l$;
- "$freundlich$": $f(c_l) = k_F c_l^{\alpha}$;
- "*langmuir*": $f(c_l) = k_L \frac{\alpha c_l}{1+\alpha c_l}$. Langmuir isotherm has been derived from thermodynamic laws. k_L denotes the maximal amount of sorbing specie which can be kept in an unit volume of a bulk matrix. Coefficient α is a fraction of adsorption and desorption rate constant $\alpha = \frac{k_a}{k_d}$.

Notation:

- Concentration in solid $c_s = \frac{n}{m_s}$ $[\mathrm{mol\,kg^{-1}}]$, where m_s, n is the mass of solid and the molar amount of the substance in unit volume, respectively;
- Concentration in liquid $c_l = \frac{m}{m_l}$ $[-]$, where m_l is the mass of liquid in unit volume. The relation between c_l and the concentration c from transport equation (2.21) is $c = c_l \varrho_l$;
- ϱ_l, ϱ_s is the liquid (solvent) density and the solid (rock) density, respectively;
- M_s denotes the molar mass of a substance;
- Multiplication parameters $k_i, i \in \{l, F, L\}$ $[\mathrm{mol\,kg^{-1}}]$ are given by `isotherm_mult`;
- Additional parameter $[\alpha] = 1$ is given by `isotherm_other`.

Denoting

$$\mu_l = \varrho_l \vartheta, \quad \mu_s = M_s \varrho_s \cdot (1-\vartheta),$$

and using (2.37), the mass balance (2.36) reduces to the equation

$$c_T = \mu_l c_l + \mu_s f(c_l), \tag{2.38}$$

which can be either solved iteratively or using interpolation. To solve (2.38) iteratively, it is very important to define the interval where to look for the solution (unknown c_l), see Figure 2.3. The lower bound is 0 (concentration can not reach negative values). The upper bound is derived using a simple mapping. Let us suppose limmited *solubility* of the selected transported substance and let us denote the limmit $\bar{c}_l$. We keep the maximal

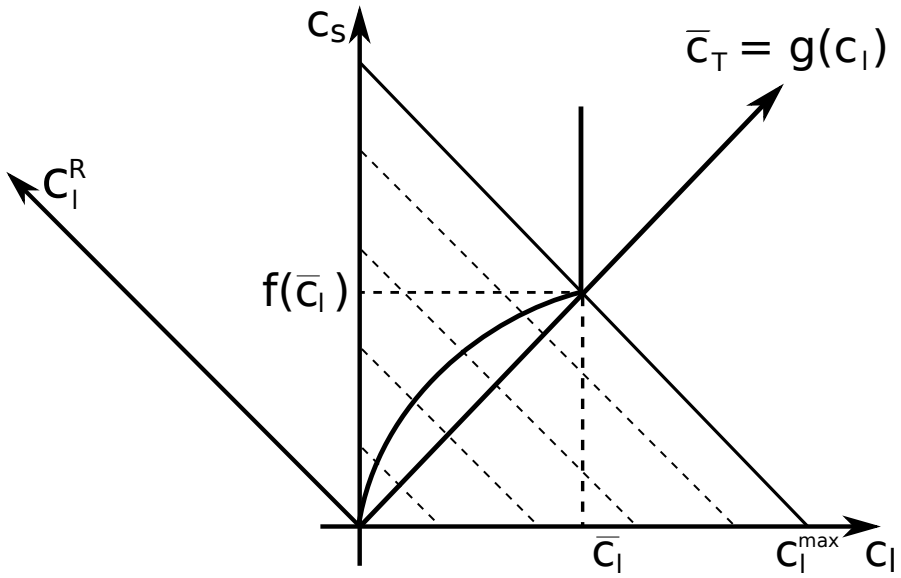


Figure 2.3: Sorption in combination with limmited solubility.

"total mass" $\bar{c}_T = \mu_l \cdot \bar{c}_l + \mu_s \cdot f(\bar{c}_l)$, but we dissolve all the mass to get maximal $c_l^{max} > \bar{c}_l$. That means $c_s = 0$ at this moment. We can slightly enlarge the interval by setting the upper bound equal to $c_l^{max} + const_{small}$.

To approximate the equation (2.38) using interpolation we need to prepare the set of values which represent $[c_l, f(c_l)]$, with c_l equidistantly distributed in transformed (rotated and rescaled) coordination system at first. The approach for construction of interpolation table follows.

1. Maximal "total mass" $\bar{c}_T = \mu_l \cdot \bar{c}_l + \mu_s \cdot f(\bar{c}_l)$ is computed.
2. Total mass step is derived $mass\_step = \bar{c}_T / n\_steps$. n_steps is given by *substeps*.
3. Appropriate $c_T^j = (mass\_step \cdot j)/\mu_l,\ j \in \{0, \ldots, n\_steps\}$ are computed.
4. The equations $\mu_l \cdot c_T^j = \mu_l \cdot c_l^j + \mu_s \cdot f(c_l^j)\ j \in \{0, \ldots, n\_steps\}$ are solved for c_l^j as unknowns. The solution is the set of ordered couples (points) $[c_l^j, f(c_l^j)],\ j \in \{0, \ldots, n\_steps\}$.

After computation of $\{[c_l^j, f(c_l^j)]\}$ we transform these coordinates to the system where the total mass is an independent variable. This is done by multiplication of precomputed points using the transformation matrix $\mathbf{A}$:

$$\begin{gathered}\vec{c}^{\,R} = \mathbf{A} \cdot \vec{c} \\ \begin{bmatrix} c_l^{R,j} \\ c_s^{R,j} \end{bmatrix} = \begin{bmatrix} \vartheta \cdot \rho_w & M_s(1-\vartheta)\rho_R \\ -M_s(1-\vartheta)\rho_R & \vartheta \cdot \rho_w \end{bmatrix} \cdot \begin{bmatrix} c_l^j \\ c_s^j \end{bmatrix} \\ j \in \{0, \ldots, n\_steps\}\end{gathered} \tag{2.39}$$

The values $c_l^{R,j}$ are equidistantly distributed and there is no reason to save them, but the values $c_s^{R,j}$ are stored in onedimensional interpolation table.

Once we have the interpolation table, we can use it for projection of $[c_l, c_s]$ transport results on the isotherm under consideration. The approach looks as folows.

1. Achieved concentrations are transformed to the coordinate system through multiplication with the matrix $\mathbf{A}$, see (2.39).

2. Transformed values are interpolated.

3. The result of interpolation is transformed back. The backward transformation consist of multiplication with $\mathbf{A}^T$ which is followed by rescaling the result. Rescaling the result is necessary because $\mathbf{A}$ is not orthonormal as it is shown bellow.

$$\mathbf{A}^T \cdot \mathbf{A} = ((\vartheta - 1)^2 \cdot M_s^2 \cdot \rho_R^2 + \vartheta^2 \cdot \rho_w^2) \cdot \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

2.5.3 Limited solubility

When $\mu_l \cdot c_l + \mu_s \cdot f(c_l) > \mu_l \cdot \bar{c}_l + \mu_s \cdot f(\bar{c}_l)$ neither iterative solver nor interpolation table is used. The aqueous concentration is set to be $\bar{c}_l$ and sorbed concentration is computed $c_s = (\mu_l \cdot c_l + \mu_s \cdot f(c_l) - \mu_l \cdot \bar{c}_l)/\mu_s$.

2.5.4 Sorption in dual porosity model

There are two parameters μ_l and μ_s, scale of aqueous concentration and scale of sorbed concentration, respectively. There is a difference in computation of these in the dual porosity model because both work on different concentrations and different zones.

Let c_{ml} and c_{ms} be concentration in liquid and in solid in the mobile zone, c_{il} and c_{is} be concentration in liquid and in solid in the immobile zone, ϑ_m and ϑ_i be the mobile and the immobile porosity, and φ be the sorbing surface.

The sorbing surface in the mobile zone is given by

$$\varphi = \frac{\vartheta_m}{\vartheta_m + \vartheta_i}, \tag{2.40}$$

while in the immobile zone it becomes

$$1 - \varphi = 1 - \frac{\vartheta_m}{\vartheta_m + \vartheta_i} = \frac{\vartheta_i}{\vartheta_m + \vartheta_i}.$$

Remind the mass balance equation (2.38). In the dual porosity model, the scaling parameters μ_l, μ_s are slightly different. In particular, the mass balance in the mobile zone reads:

$$\begin{aligned} c_T &= \mu_l \cdot c_{ml} + \mu_s \cdot c_{ms}, \\ \mu_l &= \varrho_l \cdot \vartheta_m, \\ \mu_s &= M_s \cdot \varrho_s \cdot (1 - \vartheta_m - \vartheta_i)\varphi, \end{aligned} \tag{2.41}$$

while in the immobile zone it has the form:

$$\begin{aligned} c_T &= \mu_l \cdot c_{il} + \mu_s \cdot c_{is}, \\ \mu_l &= \varrho_l \cdot \vartheta_i, \\ \mu_s &= M_s \cdot \varrho_s \cdot (1 - \vartheta_m - \vartheta_i)(1 - \varphi). \end{aligned} \tag{2.42}$$

2.6 Heat transfer

Under the assumption of thermal equilibrium between the solid and liquid phase, the energy balance equation has the form[2]

$$\partial_t \left(\delta \tilde{s} T\right) + \operatorname{div}(\varrho_l c_l T \boldsymbol{q}) - \operatorname{div}(\delta \Lambda \nabla T) = F^T + F_C^T.$$

The principal unknown is the temperature T [K]. Other quantities are:

- ϱ_l, ϱ_s [kgm^{-3}] is the density of the fluid and solid phase, respectively.
- c_l, c_s [Jkg^{-1}K^{-1}] is the heat capacity of fluid and solid phase, respectively.
- $\tilde{s}$ [Jm^{-3}K^{-1}] is the volumetric heat capacity of the porous medium defined as

  $$\tilde{s} = \vartheta \varrho_l c_l + (1 - \vartheta) \varrho_s c_s.$$

- Λ [Wm^{-1}K^{-1}] is the thermal dispersion tensor:

  $$\Lambda = \Lambda^{cond} + \Lambda^{disp}$$

  $$\Lambda^{cond} = \left(\vartheta \lambda_l^{cond} + (1 - \vartheta) \lambda_s^{cond}\right) \mathbb{I},$$

  $$\Lambda^{disp} = \vartheta \varrho_l c_l |\boldsymbol{v}| \left(\alpha_T \mathbb{I} + (\alpha_L - \alpha_T) \frac{\boldsymbol{v} \otimes \boldsymbol{v}}{|\boldsymbol{v}|^2} \right),$$

  where λ_l^{cond}, λ_s^{cond} [Wm^{-1}K^{-1}] is the thermal conductivity of the fluid and solid phase, respectively, and α_L, α_T [m] is the longitudal and transverse dispersivity in the fluid.

- F^T [Jm^{-d}s^{-1}] represents the thermal source:

  $$F^T = \delta \vartheta F_l^T + \delta (1 - \vartheta) F_s^T,$$

  $$F_l^T = f_l^T + \varrho_l c_l \sigma_l^T (T - T_l),$$
  $$F_s^T = f_s^T + \varrho_s c_s \sigma_s^T (T - T_s),$$

  where f_l^T, f_s^T [Wm^{-3}] is the density of thermal sources in fluid and solid, respectively, T_l, T_s [K] is a reference temperature and σ_l^T, σ_s^T [s^{-1}] is the heat exchange rate.

Initial and boundary conditions. At time $t = 0$ the temperature is determined by the initial condition

$$T(0, \boldsymbol{x}) = T_0(\boldsymbol{x}).$$

Given the decomposition of $\partial\Omega_d$ into $\Gamma_I \cup \Gamma_D \cup \Gamma_N \cup \Gamma_R$ (see Section 2.4), we prescribe the following boundary conditions:

- Dirichlet:

  $$T = T_D \text{ on } \Gamma_I^+ \cup \Gamma_D,$$

[2] For lower dimensions this form can be derived as in Section 2.2 using $w := \delta \tilde{s} T$, $u := T$, $\mathbb{A} := \delta \lambda \mathbb{I}$, $\boldsymbol{b} := \frac{\varrho_l c_l}{\tilde{s}} \boldsymbol{w}$.

- Homogeneous Neumann:

$$(\varrho_l c_l T\boldsymbol{q} - \delta\Lambda\nabla T)\cdot\boldsymbol{n} = 0 \text{ on } \Gamma_I^-,$$

- Neumann:

$$(\varrho_l c_l T\boldsymbol{q} - \delta\Lambda\nabla T)\cdot\boldsymbol{n} = f_N \text{ on } \Gamma_N,$$

- Robin (Newton):

$$(\varrho_l c_l T\boldsymbol{q} - \delta\Lambda\nabla T)\cdot\boldsymbol{n} = \sigma_R(T - T_D) \text{ on } \Gamma_R.$$

Communication between dimensions. Denoting T_{d+1}, T_d the temperature in Ω_{d+1} and Ω_d, respectively, the communication on the interface between Ω_{d+1} and Ω_d is described by the quantity

$$q^T_{d+1,d} = \sigma^T_{d+1,d}\frac{\delta^2_{d+1}}{\delta_d}2\Lambda_d : \boldsymbol{n}\otimes\boldsymbol{n}(T_{d+1} - T_d) + \begin{cases} \varrho_l c_l q^l_{d+1,d} T_{d+1} & \text{if } q^l_{d+1,d} \geq 0, \\ \varrho_l c_l q^l_{d+1,d}\frac{\tilde{s}_d}{\tilde{s}_{d+1}} T_d & \text{if } q^l_{d+1,d} < 0, \end{cases} \tag{2.43}$$

where

- $q^T_{d+1,d}$ [Wm^{-2}] is the density of heat flux from Ω_{d+1} to Ω_d,
- $\sigma^T_{d+1,d}$ [$-$] is a transition parameter. Its value determines the exchange of energy between dimensions due to temperature difference. In general, it is recommended to leave the default value $\sigma^T = 1$ or to set $\sigma^T = 0$ (when exchange is due to water flux only).
- $q^l_{d+1,d} = \boldsymbol{q}_{d+1}\cdot\boldsymbol{n}$ is the water flux from Ω_{d+1} to Ω_d.

The communication between dimensions is incorporated as the total flux boundary condition for the problem on Ω_{d+1}:

$$(\varrho_l c_l T\boldsymbol{q} - \delta\Lambda\nabla T)\cdot\boldsymbol{n} = q^T \tag{2.44}$$

and a source term in Ω_d:

$$F^T_{C3} = 0, \quad F^T_{C2} = q^T_{32}, \quad F^T_{C1} = q^T_{21}. \tag{2.45}$$

Energy balance. The heat equation satisfies the balance of energy in the following form:

$$e(0) + \int_0^t s(\tau)\,d\tau - \int_0^t f(\tau)\,d\tau = e(t)$$

for any instant t in the computational time interval. Here

$$e(t) := \sum_{d=1}^{3}\int_{\Omega^d}(\delta\tilde{s}T)(t,\boldsymbol{x})\,d\boldsymbol{x},$$

$$s(t) := \sum_{d=1}^{3}\int_{\Omega^d}F^T_S(t,\boldsymbol{x})\,d\boldsymbol{x},$$

$$f(t) := \sum_{d=1}^{3} \int_{\partial\Omega^d} \left(\varrho_l c_l T \boldsymbol{q} - \delta\Lambda\nabla T\right)(t, \boldsymbol{x}) \cdot \boldsymbol{n}\, d\boldsymbol{x}$$

is the energy [J], the volume source [Js^{-1}] and the energy flux [Js^{-1}] at time t, respectively. The energy, flux and source on every geometrical region is calculated at each computational time step and the values together with the control sums are written to the file `mass_balance.txt`.

Chapter 3

File formats

3.1 Main input file (CON file format)

In this section, we shall describe structure of the main input file that is given through the parameter `-s` on the command line. The file formats of other files that are referenced from the main input file and used for input of the mesh or large field data (e.g. the GMSH file format) are described in following sections. The input subsystem was designed with the aim to provide uniform initialization of C++ classes and data structures. Its structure is depicted on Figure 3.1. The structure of the input is described by the Input Types Tree (ITT) of (usually static) objects which follows the structure of the classes. The data from an input file are read by apropriate reader, their structure is checked against ITT and they are pushed into the Internal Storage Buffer (ISB). An accessor object to the root data record is the result of the file reading. The data can be retrieved through accessors which combine raw data stored in in IBS with their meaning described in ITT. ITT can be printed out in various formats providing description of the input structure both for humans and other software.

Currently, the JSON input file format is only implemented and in fact it is slight extension of the JSON file format. On the other hand the data for initialization of the C++ data structures are coded in particular way. Combination of this extension and restriction of the JSON file format produce what we call CON (C++ object notation) file format.

3.1.1 JSON for humans

Basic syntax of the CON file is very close to the JSON file format with only few extensions, namely:

- You can use C++ (or JavaScript) comments. One line comments `//` and multi-line comments `/* */`.
- The quoting of the keys is optional if they do not contain spaces (holds for all CON keys).
- You can use equality sign `=` instead of colon `:` for separation of keys and values in JSON objects.

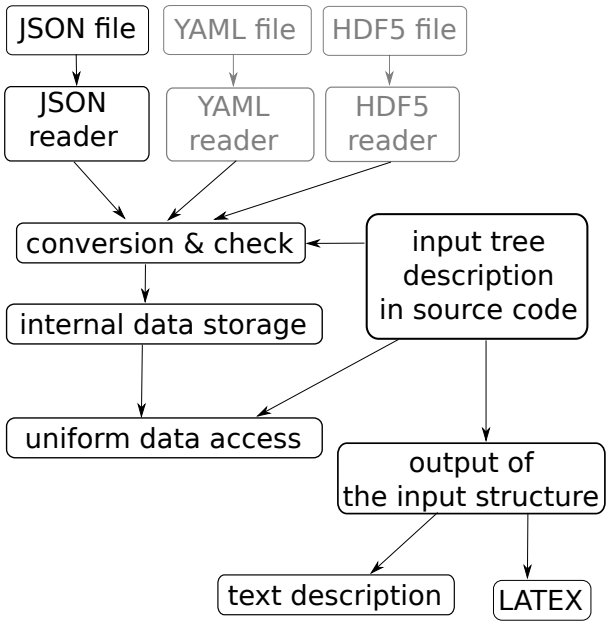


Figure 3.1: Sturucture of the input subsystem. Grey boxes are not implemented yet.

- You can use any whitespace to separate tokens in JSON object or JSON array.

The aim of these extensions is to simplify writing input files manually. However these extensions can be easily filtered out and converted to the generic JSON format. For the description of the JSON format we refer to `http://www.json.org/`.

3.1.2 CON constructs

The CON file format constructs are designed for initialization of C++ strongly typed variables. The primitive data types can be initialized from the primitive CON constructs:

- *Bool* — initialized from the JSON keywords `true` and `false`.
- *Double*, *Integer* — initialized from JSON numeric data.
- *String*, *FileName*, *Selections* — initialized from JSON strings

Selections are typed like the C++ enum types that are initialized from them. Various kind of containers can be initialized by the *Array* construct, that is an JSON array with elements of the same CON type. The C++ structures and classes can be initialize from the *Record* construct, which is represented by a JSON object. However, in constrast to JSON, these Records have different types in similar way as the strong typed C++ structures. The types are described by ITT of the particular program which can be printed out in several formats, in particular description of ITT for Flow123d forms content of Chapter 4. In order to allow certain kind of polymorphism, we introduce also the *AbstractRecord* construct, where the type of the record is not given by ITT but can be chosen as part of the input.

3.1.3 CON special keys

All keys in Records should be in lower case, possibly using digits and underscore. The keys all in upper case are reserved for special function in the CON file. These are:

TYPE key :

```
TYPE=<Selection of AbstractRecord>
```

Is used to specify particular type of an AbstractRecord. This way you can choose which particular implementation of an abstract C++ class should be instantiated. The value of the key is a string from the Selection that consists of names of Records that was declared as descendants of the AbstractRecord.

REF key :

```
{ REF=<address> }
```

The record in input file that contains only the key `REF` is replaced by the JSON entity that is referenced by the `<address>`. The address is a string with format similar to UNIX path, i.e. with grammar

```
<address> = <address> / <item>
          = <item>
          = <null>
<item> = <index>
       = <key>
       = ..
```

where `index` is non-negative integer and key is valid CON record key (lowercase, digits, underscores). The address can be absolute or relative identification of an entity. The relative address is relative to the entity in which the reference record is contained. One can use two dots `".."` to move to parent entity.

Example:

```
mesh={
        file_name="xyz"
}
array=[
        {x=1 y=0}
        {x=2 y=0}
        {x=3 y=0}
]
outer_record={
        output_file="x_out"
        inner_record={
                output_file={REF="../output_file"} // value "x_out"
        }
        x={REF="/array/2/x"}                       // value "3"
        f_name={REF="/mesh/file_name"}             // value "xyz"
}
```

3.1.4 Record types

A Record type is given by the set of key specifications, which in turn consist from: key name, type of value and default value specification. Default value specification can be:

obligatory — means no default value, which has to be specified at input.

optional — means no default value, but value is needs not to be specified. Unspecified value usually means that you turn off some functionality.

default at declaration — the default value is explicitly given in declaration and is automatically provided by the input subsystem if needed

default at read time — the default value is provided at read time, usually from some other variable. In the documentation, there is only textual description where the default value comes from.

Implicit creation of composed entities

Consider a Record type in which all keys have default values (possibly except one). Then the specification of the Record can contain a *key for default construction*. User can specify only the value of this particular key instead of the whole record, all other keys are initialized from its default values. Moreover, an AbstractRecord type may have a default value for the `TYPE` key. This allows to express simple tasks by simple inputs but still make complex inputs possible. Similar functionality holds for arrays. If the user sets a non-array value where an array is expected the reader provides an array with a unique element holding the given value.

3.2 Important Record types of Flow123d input

3.2.1 Mesh record

The mesh record provides initialization for the computational mesh consisting of points, lines, triangles and tetrahedrons in 3D space. Currently, we support only GMSH mesh file format MSH ASCII. The input file is provided by the key `mesh_file`. The file format allows to group elements into *regions* identified either by ID number or by string label. The regions with labels starting with the dot character are treated as *boundary regions*. Their elements are removed from the computational domain, however they can be used to specify boundary conditions. Other regions are called *bulk regions*. User can create new labeled regions through the key `regions`, the new region can be specified either by its ID or by list of IDs of its elements. The latter possibility overrides original region assigned to the elements, which can be useful for specification of small areas "ad hoc". The key `sets` allows specification of sets of regions in terms of list of region IDs or labels and basic set operations. The difference between regions and sets is that regions form disjoint covering of elements, the sets, however, may overlap. There are three predefined region sets: "ALL", "BOUNDARY", "BULK".

3.2.2 Field records

A general time and space dependent, scalar, vector, or tensor valued function can be specified through the family of abstract records Field $R^m -> \mathcal{S}$, where m is currently always $m = 3$ and $\mathcal{S}$ is a specification of the target space, which can be:

- $\mathcal{T}$ — scalar valued field, with scalars of type $\mathcal{T}$
- $\mathcal{T}[d]$ — vector valued field, with vector of fixed size d and elements of type $\mathcal{T}$
- $\mathcal{T}[\mathtt{n}]$ — vector valued field, with vector of variable size (given by some input) and elements of type $\mathcal{T}$
- $\mathcal{T}[d, d]$ — tensor valued field, with square tensor of fixed size and elements of type $\mathcal{T}$

the scalar types can be

- **Real** — scalar real valued field
- **Int** — scalar integer valued field
- **Enum** — scalar non negative integer valued field, should be convertible to appropriate C++ enum type

Each of these abstract record has the same set of descendants which implement various algorithms to specify and compute values of the field. These are

FieldConstant — field that is constant in space

FieldFormula — field that is given by runtime parsed formula using x, y, z, t coordinates. The Function Parser library is used with syntax rules described here.

FieldPython — field can be implemented by Python script either specified by string (key `script_string`) or in external file (key `script_file`.

FieldElementwise — discrete field, currently only piecewise constant field on elements is supported, the field can given by the MSH ASCII file specified in key `gmsh_file` and field name in the file given by key `field_name`. The file must contain same mesh as is used for computation.

FieldInterpolated — allows interpolation between different meshes. Not yet fully supported.

Several automatic conversions are implemented. Scalar values can be used to set constant vectors or tensors. Vector value of size d can be used to set diagonal tensor $d \times d$. Vector value of size $d(d-1)/2$, e.g. 6 for $d = 3$, can be used to set symmetric tensor. These rules apply only for FieldConstant and FieldFormula. Moreover, all Field abstract types have default value `TYPE=FieldConstant`. Thus you can just use the constant value instead of the whole record.

Examples:

```
constant_scalar_function = 1.0
// is same as
constant_scalar_function = {
  TYPE=FieldConstant,
  value=1.0
}

conductivity_tensor = [1 ,2, 3]
// is same as
conductivity_tensor = {
    TYPE=FieldConstant,
    value=[[1,0,0],[0,2,0],[0,0,3]]
}

concentration = {
    TYPE=FieldFormula,
    value="x+y+z"
}
//is same as (provided the vector has 2 elements)
concentration = {
    TYPE=FieldFormula,
    value=["x+y+z", "x+y+z"]
}
```

3.2.3 Field data for equations

Every equation record has key `input_fields`, intended to set both the bulk and boundary fields. These keys contain an array of region-time initialization records like the `Data` record of the DarcyFlow equation. Every such record specifies fields on particular region (keys `region` and `rid`) or on a region set (key `r_set`) starting from the time specified by the key `time`. The array is processed sequentially and latter values overwrite the previous ones. Times should form a non-decreasing sequence.

Example:

```
input_fields = [
    { // time=0.0  - default value
        r_set="BULK",
        conductivity=1   // setting the conductivity field on all regions
    },
    {
        region="2d_part",
        conductivity=2  // overwriting the previous value
    },
    {   time=1.0,
        region="2d_part",
        conductivity={
            // from time=1.0 we switch to the linear function in time
            TYPE=FieldFormula,
```

```
            value="2+t"
        }
    },
    {   time=2.0,
        region="2d_part",
        conductivity={
            // from time=2.0 we switch to elementwise field, but only
            // on the region "2d_part"
            TYPE=FieldElementwise,
            gmsh_file="./input/data.msh",
            field_name="conductivity"
        }
    }
]
```

3.3 Mesh and data file format MSH ASCII

Currently, the only supported format for the computational mesh is MSH ASCII format used by the GMSH software. You can find its documentation on:

http://geuz.org/gmsh/doc/texinfo/gmsh.html#MSH-ASCII-file-format

The scheme of the file is as follows:

```
$MeshFormat
<format version>
$EndMeshFormat

$PhysicalNames
<number of items>
<dimension>     <region ID>     <region label>
...
$EndPhysicalNames

$Nodes
<number of nodes>
<node ID> <X coord> <Y coord> <Z coord>
...
$EndNodes

$Elements
<number of elements>
<element ID> <element shape> <n of tags> <tags> <nodes>
...
$EndElements

$ElementData
<n of string tags>
    <field name>
```

```
    <interpolation scheme>
<n of double tags>
    <time>
<n of integer tags>
    <time step index>
    <n of components>
    <n of items>
    <partition index>
<element ID> <component 1> <component 2> ...
...
$EndElementData
```

Detailed description of individual sections:

`PhysicalNames` : Assign labels to region IDs. Elements of one region should have common dimension. Flow123d interprets regions with labels starting with period as the boundary elements that are not used for calculations.

`Nodes` : `<number of nodes>` is also number of data lines that follows. Node IDs are unique but need not to form an aritmetic sequance. Coordinates are float numbers.

`Elements` : Element IDs are unique but need not to form an aritmetic sequence. Integer code `<element shape>` represents the shape of element, we support only points (15), lines (1), triangles (2), and tetrahedrons (4). Default number of tags is 3. The first is the region ID, the second is ID of the geometrical entity (that was used in original geometry file from which the mesh was generated), and the third tag is the partition number. `nodes` is list of node IDs with size according to the element shape.

`ElementData` : The header has 2 string tags, 1 double tag, and 4 integer tags with default meaning. For the purpose of the `FieldElementwise` the tags `<field name>`, `<n of components>`, and `<n of items>` are obligatory. This header is folowed by field data on individual elements. Flow123d assumes that elements are sorted by `element ID`, but doesn't need to form a continuos sequence.

3.4 Output files

Flow123d supports output of scalar, vector and tensor data fields into two formats. The first is the native format of the GMSH software (usually with extension `msh`) which contains computational mesh followed by data fields for sequence of time levels. The second is the XML version of VTK files. These files can be viewed and post-processed by several visualization software packages. However, our primal goal is to support data transfer into the Paraview visualization software. See key `format`.

Input record of every equation (flow, transport, reactions, heat) contains the keys `output_stream` and `output_fields`. In `output_stream`, the name and type of the output file is specified. Further, in `output_fields`, one determines the list of fields intended for output. The available output fields include input data as well as the simulation results.

Below we mention the most important output fields of all equations and link to the complete lists.

Darcy flow	
`pressure_p0`	Pressure head [m], piecewise constant on every element. This field is directly produced by the MH method and thus contains no postprocessing error.
`pressure_p1`	Same pressure head field, but interpolated into $P1$ continuous scalar field. Namely you lost dicontinuities on fractures.
`velocity_p0`	Vector field of water flux [m^3s^{-1}]. For every element we evaluate discrete flux field in barycenter.
`piezo_head_p0`	Piezometric head [m], piecewise constant on every element. This is just pressure on element plus z-coordinate of the barycenter. This field is produced only on demand (see key `piezo_head_p0`).
complete list	See Darcy flow output fields.
Convection transport	
`conc`	Concentration [kgm^{-3}], piecewise constant on every element.
complete list	See Convection transport output fields.
Transport with dispersion	
`conc`	Concentration [kgm^{-3}], piecewise linear on every element. Even if higher order polynomial approximation is used in simulation, the results are saved only in element corners.
complete list	See Transport with dispersion output fields.
Dual porosity	
`conc_immobile`	Concentration [kgm^{-3}] in immobile zone, piecewise linear on every element.
complete list	See Dual porosity output fields.
Sorption, Mobile sorption, Immobile sorption	
`conc_solid`	Concentration [$mol\,kg^{-1}$] of sorbed substance, piecewise linear on every element.
complete list	See Sorption output fields, Mobile sorption output fields, Immobile sorption output fields.
Heat transfer	
`temperature`	Temperature [K], piecewise linear on every element. Even if higher order polynomial approximation is used in simulation, the results are saved only in element corners.
complete list	See Heat transfer output fields.

3.4.1 Auxiliary output files

Profiling information

On every run we collect some basic profiling informations. After all computations these data are written into the file `profiler%y%m%d_%H.%M.%S.out` where `%y`, `%m`, `%d`, `%H`, `%M`, `%S` are two digit numbers representing year, month, day, hour, minute, and second of the program start time.

Water flux information

File contains water flow balance, total inflow and outflow over boundary segments. Further there is total water income due to sources (if they are present).

Raw water flow data file

You can force Flow123d to write raw data about results of MH method. The file format is:

```
$FlowField
T=<time>
<number fo elements>
<eid> <pressure> <flux x> <flux y> <flux z> <number of sides> <pressures on sides> <flu
...
$EndFlowField
```

where

`<time>` — is simulation time of the raw output.

`<number of elements>` — is number of elements in mesh, which is same as number of subsequent lines.

`<eid>` — element id same as in the input mesh.

`<flux x,y,z>` — components of water flux interpolated to barycenter of the element

`<number of sides>` — number of sides of the element, infulence number of remaining values

`<pressures on sides>` — for every side average of the pressure over the side

`<fluxes on sides>` — for ever side total flux through the side

Chapter 4

Main input file reference

Application constructor

---

record: **Root**

Root record of JSON input for Flow123d.

`problem` = <*abstract type: Problem*>

Default: <*obligatory*>

Simulation problem to be solved.

`pause_after_run` = <*Bool*>

Default: false

If true, the program will wait for key press before it terminates.

---

abstract type: **Problem**

Descendants:

The root record of description of particular the problem to solve.

`SequentialCoupling`

---

record: **SequentialCoupling** implements abstract type: Problem

---

Record with data for a general sequential coupling.

`TYPE` = <*selection: Problem_TYPE_selection*>

Default: SequentialCoupling

Sub-record selection.

`description` = <*String (generic)*>

Default: <*optional*>

Short description of the solved problem.
Is displayed in the main log, and possibly in other text output files.

`mesh` = <*record: Mesh*>

Default: <*obligatory*>

Computational mesh common to all equations.

`time` = <*record:* *TimeGovernor*>

Default: <*optional*>

Simulation time frame and time step.

`primary_equation` = <*abstract type:* *DarcyFlowMH*>

Default: <*obligatory*>

Primary equation, have all data given.

`secondary_equation` = <*abstract type:* *Transport*>

Default: <*optional*>

The equation that depends (the velocity field) on the result of the primary equation.

---

record: **Mesh**

Record with mesh related data.

`mesh_file` = <*input file name*>

Default: <*obligatory*>

Input file with mesh description.

`regions` = <*Array of record:* *Region*>

Default: <*optional*>

List of additional region definitions not contained in the mesh.

`sets` = <*Array of record:* *RegionSet*>

Default: <*optional*>

List of region set definitions. There are three region sets implicitly defined: ALL (all regions of the mesh), BOUNDARY (all boundary regions), and BULK (all bulk regions)

`partitioning` = <*record:* *Partition*>

Default: any_neighboring

Parameters of mesh partitioning algorithms.

---

record: **Region**

Definition of region of elements.

`name` = <*String (generic)*>

Default: <*obligatory*>

Label (name) of the region. Has to be unique in one mesh.

`id` = <*Integer [0,]*>

Default: <*obligatory*>

The ID of the region to which you assign label.

`element_list` = <*Array of Integer [0,]*>

Default: <*optional*>

Specification of the region by the list of elements. This is not recomended

---

record: **RegionSet**

Definition of one region set.

`name` = <*String (generic)*>

Default: <*obligatory*>

Unique name of the region set.

`region_ids` = <*Array of Integer [0,]*>

Default: <*optional*>

List of region ID numbers that has to be added to the region set.

`region_labels` = <*Array of String (generic)*>

Default: <*optional*>

List of labels of the regions that has to be added to the region set.

`union` = <*Array [2, 2] of String (generic)*>

Default: <*optional*>

Defines region set as a union of given pair of sets. Overrides previous keys.

`intersection` = <*Array [2, 2] of String (generic)*>

Default: <*optional*>

Defines region set as an intersection of given pair of sets. Overrides previous keys.

`difference` = <*Array [2, 2] of String (generic)*>

Default: <*optional*>

Defines region set as a difference of given pair of sets. Overrides previous keys.

---

record: **Partition** constructible from key: graph_type

Setting for various types of mesh partitioning.

`tool` = <*selection: PartTool*>

Default: METIS

Software package used for partitioning. See corresponding selection.

`graph_type` = <*selection: GraphType*>

Default: any_neighboring

Algorithm for generating graph and its weights from a multidimensional mesh.

---

selection type: **PartTool**

Select the partitioning tool to use.

Possible values:

`PETSc` : Use PETSc interface to various partitioning tools.

`METIS` : Use direct interface to Metis.

---

selection type: **GraphType**

Different algorithms to make the sparse graph with weighted edges from the multidimensional mesh. Main difference is dealing with neighborings of elements of different dimension.

Possible values:

`any_neighboring` : Add edge for any pair of neighboring elements.

`any_wight_lower_dim_cuts` : Same as before and assign higher weight to cuts of lower dimension in order to make them stick to one face.

`same_dimension_neghboring` : Add edge for any pair of neighboring elements of same dimension (bad for matrix multiply).

---

record: **TimeGovernor** constructible from key: init_dt

Setting of the simulation time. (can be specific to one equation)

`start_time` = *<Double >*

Default: 0.0

Start time of the simulation.

`end_time` = *<Double >*

Default: *<optional>*

End time of the simulation.

`init_dt` = *<Double [0,]>*

Default: 0.0

Initial guess for the time step.
The time step is fixed if the hard time step limits are not set.
If set to 0.0, the time step is determined in fully autonomous way if the equation supports it.

`min_dt` = *<Double [0,]>*

Default: "Machine precision or 'init_dt' if specified"

Hard lower limit for the time step.

`max_dt` = *<Double [0,]>*

Default: "Whole time of the simulation or 'init_dt' if specified"

Hard upper limit for the time step.

---

abstract type: **DarcyFlowMH**

Descendants:

Mixed-Hybrid solver for saturated Darcy flow.

`Steady_MH`

`Unsteady_MH`

`Unsteady_LMH`

---

record: **Steady_MH** implements abstract type: DarcyFlowMH

Mixed-Hybrid solver for STEADY saturated Darcy flow.

`TYPE` = *<selection: DarcyFlowMH_TYPE_selection>*

Default: Steady_MH

Sub-record selection.

`n_schurs` = *<Integer [0, 2]>*

Default: 2

Number of Schur complements to perform when solving MH sytem.

`solver` = *<abstract type: LinSys>*

Default: *<obligatory>*

Linear solver for MH problem.

`output` = *<record: DarcyMHOutput>*

Default: *<obligatory>*

Parameters of output form MH module.

`mortar_method` = *<selection: MH_MortarMethod>*

Default: None

Method for coupling Darcy flow between dimensions.

`input_fields` = *<Array of record: DarcyFlowMH_Data>*

Default: *<obligatory>*

---

abstract type: **LinSys**

Descendants:

Linear solver setting.

`Petsc`

Bddc

---

record: **Petsc** implements abstract type: LinSys

Solver setting.

`TYPE` = *<selection: LinSys_TYPE_selection>*

Default: Petsc

Sub-record selection.

`r_tol` = *<Double [0, 1]>*

Default: 1.0e-7

Relative residual tolerance (to initial error).

`max_it` = *<Integer [0,]>*

Default: 10000

Maximum number of outer iterations of the linear solver.

`a_tol` = *<Double [0,]>*

Default: 1.0e-9

Absolute residual tolerance.

`options` = *<String (generic)>*

Default:

Options passed to PETSC before creating KSP instead of default setting.

---

record: **Bddc** implements abstract type: LinSys

Solver setting.

`TYPE` = *<selection: LinSys_TYPE_selection>*

Default: Bddc

Sub-record selection.

`r_tol` = *<Double [0, 1]>*

Default: 1.0e-7

Relative residual tolerance (to initial error).

`max_it` = *<Integer [0,]>*

Default: 10000

Maximum number of outer iterations of the linear solver.

`max_nondecr_it` = *<Integer [0,]>*

Default: 30

Maximum number of iterations of the linear solver with non-decreasing residual.

`number_of_levels` = *<Integer [0,]>*

Default: 2

Number of levels in the multilevel method (=2 for the standard BDDC).

`use_adaptive_bddc` = <*Bool*>

Default: false

Use adaptive selection of constraints in BDDCML.

`bddcml_verbosity_level` = <*Integer [0, 2]*>

Default: 0

Level of verbosity of the BDDCML library: 0 - no output, 1 - mild output, 2 - detailed output.

---

record: **DarcyMHOutput**

Parameters of MH output.

`output_stream` = <*record: OutputStream*>

Default: <*obligatory*>

Parameters of output stream.

`output_fields` = <*Array of selection: DarcyMHOutput_Selection*>

Default: <*obligatory*>

List of fields to write to output file.

`balance_output` = <*output file name*>

Default: water_balance.txt

Output file for water balance table.

`compute_errors` = <*Bool*>

Default: false

SPECIAL PURPOSE. Computing errors pro non-compatible coupling.

`raw_flow_output` = <*output file name*>

Default: <*optional*>

Output file with raw data form MH module.

---

record: **OutputStream**

Parameters of output.

`file` = <*output file name*>

Default: <*obligatory*>

File path to the connected output file.

`format` = <*abstract type: OutputTime*>

Default: <*optional*>

Format of output stream and possible parameters.

`time_step` = *<Double [0,]>*

Default: *<optional>*

Time interval between outputs.
Regular grid of output time points starts at the initial time of the equation and ends at the end time which must be specified.
The start time and the end time are always added.

`time_list` = *<Array of Double [0,]>*

Default: *<optional>*

Explicit array of output time points (can be combined with 'time_step'.

`add_input_times` = *<Bool>*

Default: false

Add all input time points of the equation, mentioned in the 'input_fields' list, also as the output points.

---

abstract type: **OutputTime**

Descendants:

Format of output stream and possible parameters.

vtk

gmsh

---

record: **vtk** implements abstract type: OutputTime

Parameters of vtk output format.

`TYPE` = *<selection: OutputTime_TYPE_selection>*

Default: vtk

Sub-record selection.

`variant` = *<selection: VTK variant (ascii or binary)>*

Default: ascii

Variant of output stream file format.

`parallel` = *<Bool>*

Default: false

Parallel or serial version of file format.

`compression` = *<selection: Type of compression of VTK file format>*

Default: none

Compression used in output stream file format.

---

selection type: **VTK variant (ascii or binary)**

Possible values:

`ascii` : ASCII variant of VTK file format

`binary` : Binary variant of VTK file format (not supported yet)

selection type: **Type of compression of VTK file format**

Possible values:

`none` : Data in VTK file format are not compressed

`zlib` : Data in VTK file format are compressed using zlib (not supported yet)

record: **gmsh** implements abstract type: OutputTime

Parameters of gmsh output format.

`TYPE` = *<selection: OutputTime_TYPE_selection>*

Default: gmsh

Sub-record selection.

selection type: **DarcyMHOutput_Selection**

Selection of fields available for output.

Possible values:

`anisotropy` : Output of the field anisotropy (Anisotropy of the conductivity tensor.).

`cross_section` : Output of the field cross_section (Complement dimension parameter (cross section for 1D, thickness for 2D).).

`conductivity` : Output of the field conductivity (Isotropic conductivity scalar.).

`sigma` : Output of the field sigma (Transition coefficient between dimensions.).

`water_source_density` : Output of the field water_source_density (Water source density.).

`init_pressure` : Output of the field init_pressure (Initial condition as pressure).

`storativity` : Output of the field storativity (Storativity.).

`pressure_p0` : Output of the field pressure_p0.

`pressure_p1` : Output of the field pressure_p1.

`piezo_head_p0` : Output of the field piezo_head_p0.

`velocity_p0` : Output of the field velocity_p0.

`subdomain` : Output of the field subdomain.

`pressure_diff` : Output of the field pressure_diff.

`velocity_diff` : Output of the field velocity_diff.

`div_diff` : Output of the field div_diff.

---

selection type: **MH_MortarMethod**

Possible values:

`None` : Mortar space: P0 on elements of lower dimension.

`P0` : Mortar space: P0 on elements of lower dimension.

`P1` : Mortar space: P1 on intersections, using non-conforming pressures.

---

record: **DarcyFlowMH_Data**

Record to set fields of the equation. The fields are set only on the domain specified by one of the keys: 'region', 'rid', 'r_set' and after the time given by the key 'time'. The field setting can be overridden by any DarcyFlowMH_Data record that comes later in the boundary data array.

`r_set` = *<String (generic)>*

Default: *<optional>*

Name of region set where to set fields.

`region` = *<String (generic)>*

Default: *<optional>*

Label of the region where to set fields.

`rid` = *<Integer [0,]>*

Default: *<optional>*

ID of the region where to set fields.

`time` = *<Double [0,]>*

Default: 0.0

Apply field setting in this record after this time.
These times have to form an increasing sequence.

`anisotropy` = *<abstract type: Field:R3 → Real[3,3]>*

Default: *<optional>*

Anisotropy of the conductivity tensor.

`cross_section` = *<abstract type: Field:R3 → Real>*

Default: *<optional>*

Complement dimension parameter (cross section for 1D, thickness for 2D).

`conductivity` = *<abstract type: Field:R3 → Real>*

Default: *<optional>*

Isotropic conductivity scalar.

`sigma` = *<abstract type: Field:R3 → Real>*

Default: <*optional*>

Transition coefficient between dimensions.

`water_source_density` = <*abstract type: Field:R3 → Real*>

Default: <*optional*>

Water source density.

`bc_type` = <*abstract type: Field:R3 → Enum*>

Default: <*optional*>

Boundary condition type, possible values:

`bc_pressure` = <*abstract type: Field:R3 → Real*>

Default: <*optional*>

Dirichlet BC condition value for pressure.

`bc_flux` = <*abstract type: Field:R3 → Real*>

Default: <*optional*>

Flux in Neumman or Robin boundary condition.

`bc_robin_sigma` = <*abstract type: Field:R3 → Real*>

Default: <*optional*>

Conductivity coefficient in Robin boundary condition.

`init_pressure` = <*abstract type: Field:R3 → Real*>

Default: <*optional*>

Initial condition as pressure

`storativity` = <*abstract type: Field:R3 → Real*>

Default: <*optional*>

Storativity.

`bc_piezo_head` = <*abstract type: Field:R3 → Real*>

Default: <*optional*>

Boundary condition for pressure as piezometric head.

`init_piezo_head` = <*abstract type: Field:R3 → Real*>

Default: <*optional*>

Initial condition for pressure as piezometric head.

`flow_old_bcd_file` = <*input file name*>

Default: <*optional*>

File with mesh dependent boundary conditions (obsolete).

---

abstract type: **Field:R3 → Real[3,3]** default descendant: FieldConstant

---

Descendants:

Abstract record for all time-space functions.

FieldConstant

FieldPython

FieldFormula

FieldElementwise

FieldInterpolatedP0

---

record: **FieldConstant** implements abstract type: Field:R3 → Real[3,3] constructible from key: value

---

R3 → Real[3,3] Field constant in space.

TYPE = <*selection: Field:R3 → Real[3,3]_TYPE_selection*>

Default: FieldConstant

Sub-record selection.

value = <*Array [1,] of Array [1,] of Double* >

Default: <*obligatory*>

Value of the constant field.
For vector values, you can use scalar value to enter constant vector.
For square NxN-matrix values, you can use:
vector of size N to enter diagonal matrix
vector of size (N+1)*N/2 to enter symmetric matrix (upper triangle, row by row)
scalar to enter multiple of the unit matrix.

---

record: **FieldPython** implements abstract type: Field:R3 → Real[3,3]

R3 → Real[3,3] Field given by a Python script.

TYPE = <*selection: Field:R3 → Real[3,3]_TYPE_selection*>

Default: FieldPython

Sub-record selection.

script_string = <*String (generic)*>

Default: "Obligatory if 'script_file' is not given."

Python script given as in place string

script_file = <*input file name*>

Default: "Obligatory if 'script_striong' is not given."

Python script given as external file

function = <*String (generic)*>

Default: <*obligatory*>

Function in the given script that returns tuple containing components of the return type.

For NxM tensor values: tensor(row,col) = tuple(M*row + col).

---

record: **FieldFormula** implements abstract type: Field:R3 → Real[3,3]

R3 → Real[3,3] Field given by runtime interpreted formula.

`TYPE` = *<selection: Field:R3 → Real[3,3]_TYPE_selection>*

Default: FieldFormula

Sub-record selection.

`value` = *<Array [1,] of Array [1,] of String (generic)>*

Default: *<obligatory>*

String, array of strings, or matrix of strings with formulas for individual entries of scalar, vector, or tensor value respectively.
For vector values, you can use just one string to enter homogeneous vector.
For square NxN-matrix values, you can use:
array of strings of size N to enter diagonal matrix
array of strings of size (N+1)*N/2 to enter symmetric matrix (upper triangle, row by row)
just one string to enter (spatially variable) multiple of the unit matrix.
Formula can contain variables x,y,z,t and usual operators and functions.

---

record: **FieldElementwise** implements abstract type: Field:R3 → Real[3,3]

R3 → Real[3,3] Field constant in space.

`TYPE` = *<selection: Field:R3 → Real[3,3]_TYPE_selection>*

Default: FieldElementwise

Sub-record selection.

`gmsh_file` = *<input file name>*

Default: *<obligatory>*

Input file with ASCII GMSH file format.

`field_name` = *<String (generic)>*

Default: *<obligatory>*

The values of the Field are read from the $ElementData section with field name given by this key.

---

record: **FieldInterpolatedP0** implements abstract type: Field:R3 → Real[3,3]

R3 → Real[3,3] Field constant in space.

`TYPE` = *<selection: Field:R3 → Real[3,3]_TYPE_selection>*

Default: FieldInterpolatedP0

Sub-record selection.

`gmsh_file` = *<input file name>*

Default: *<obligatory>*

Input file with ASCII GMSH file format.

`field_name` = *<String (generic)>*

Default: *<obligatory>*

The values of the Field are read from the $ElementData section with field name given by this key.

---

abstract type: **Field:R3 → Real** default descendant: FieldConstant

---

Descendants:

Abstract record for all time-space functions.

`FieldConstant`

`FieldPython`

`FieldFormula`

`FieldElementwise`

`FieldInterpolatedP0`

---

record: **FieldConstant** implements abstract type: Field:R3 → Real constructible from key: value

---

R3 → Real Field constant in space.

`TYPE` = *<selection: Field:R3 → Real_TYPE_selection>*

Default: FieldConstant

Sub-record selection.

`value` = *<Double >*

Default: *<obligatory>*

Value of the constant field.
For vector values, you can use scalar value to enter constant vector.
For square NxN-matrix values, you can use:
vector of size N to enter diagonal matrix
vector of size (N+1)*N/2 to enter symmetric matrix (upper triangle, row by row)
scalar to enter multiple of the unit matrix.

---

record: **FieldPython** implements abstract type: Field:R3 → Real

---

R3 → Real Field given by a Python script.

`TYPE` = *<selection: Field:R3 → Real_TYPE_selection>*

Default: FieldPython

Sub-record selection.

`script_string` = *<String (generic)>*

Default: "Obligatory if 'script_file' is not given."

Python script given as in place string

`script_file` = *<input file name>*

Default: "Obligatory if 'script_striong' is not given."

Python script given as external file

`function` = *<String (generic)>*

Default: *<obligatory>*

Function in the given script that returns tuple containing components of the return type.
For NxM tensor values: tensor(row,col) = tuple(M*row + col).

---

record: **FieldFormula** implements abstract type: Field:R3 → Real

---

R3 → Real Field given by runtime interpreted formula.

`TYPE` = *<selection: Field:R3 → Real_TYPE_selection>*

Default: FieldFormula

Sub-record selection.

`value` = *<String (generic)>*

Default: *<obligatory>*

String, array of strings, or matrix of strings with formulas for individual entries of scalar, vector, or tensor value respectively.
For vector values, you can use just one string to enter homogeneous vector.
For square NxN-matrix values, you can use:
array of strings of size N to enter diagonal matrix
array of strings of size (N+1)*N/2 to enter symmetric matrix (upper triangle, row by row)
just one string to enter (spatially variable) multiple of the unit matrix.
Formula can contain variables x,y,z,t and usual operators and functions.

---

record: **FieldElementwise** implements abstract type: Field:R3 → Real

---

R3 → Real Field constant in space.

`TYPE` = *<selection: Field:R3 → Real_TYPE_selection>*

Default: FieldElementwise

Sub-record selection.

`gmsh_file` = *<input file name>*

Default: *<obligatory>*

Input file with ASCII GMSH file format.

`field_name` = *<String (generic)>*

Default: *<obligatory>*

The values of the Field are read from the $ElementData section with field name given by this key.

---

record: **FieldInterpolatedP0** implements abstract type: Field:R3 → Real

---

R3 → Real Field constant in space.

`TYPE` = *<selection: Field:R3 → Real_TYPE_selection>*

Default: FieldInterpolatedP0

Sub-record selection.

`gmsh_file` = *<input file name>*

Default: *<obligatory>*

Input file with ASCII GMSH file format.

`field_name` = *<String (generic)>*

Default: *<obligatory>*

The values of the Field are read from the $ElementData section with field name given by this key.

---

abstract type: **Field:R3 → Enum** default descendant: FieldConstant

---

Descendants:

Abstract record for all time-space functions.

`FieldConstant`

`FieldFormula`

`FieldPython`

`FieldInterpolatedP0`

`FieldElementwise`

---

record: **FieldConstant** implements abstract type: Field:R3 → Enum constructible from key: value

---

R3 → Enum Field constant in space.

`TYPE` = *<selection: Field:R3 → Enum_TYPE_selection>*

Default: FieldConstant

Sub-record selection.

`value` = *<selection: EqData_bc_Type>*

Default: *<obligatory>*

Value of the constant field.
For vector values, you can use scalar value to enter constant vector.

For square NxN-matrix values, you can use:
vector of size N to enter diagonal matrix
vector of size (N+1)*N/2 to enter symmetric matrix (upper triangle, row by row)
scalar to enter multiple of the unit matrix.

---

selection type: **EqData_bc_Type**

Possible values:

`none` : Homogeneous Neoumann BC.

`dirichlet` :

`neumann` :

`robin` :

`total_flux` :

---

record: **FieldFormula** implements abstract type: Field:R3 → Enum

---

R3 → Enum Field given by runtime interpreted formula.

`TYPE` = *<selection: Field:R3 → Enum_TYPE_selection>*

Default: FieldFormula

Sub-record selection.

`value` = *<String (generic)>*

Default: *<obligatory>*

String, array of strings, or matrix of strings with formulas for individual entries of scalar, vector, or tensor value respectively.
For vector values, you can use just one string to enter homogeneous vector.
For square NxN-matrix values, you can use:
array of strings of size N to enter diagonal matrix
array of strings of size (N+1)*N/2 to enter symmetric matrix (upper triangle, row by row)
just one string to enter (spatially variable) multiple of the unit matrix.
Formula can contain variables x,y,z,t and usual operators and functions.

---

record: **FieldPython** implements abstract type: Field:R3 → Enum

---

R3 → Enum Field given by a Python script.

`TYPE` = *<selection: Field:R3 → Enum_TYPE_selection>*

Default: FieldPython

Sub-record selection.

`script_string` = *<String (generic)>*

Default: "Obligatory if 'script_file' is not given."

Python script given as in place string

`script_file` = *<input file name>*

Default: "Obligatory if 'script_striong' is not given."

Python script given as external file

`function` = *<String (generic)>*

Default: *<obligatory>*

Function in the given script that returns tuple containing components of the return type.
For NxM tensor values: tensor(row,col) = tuple(M*row + col).

---

record: **FieldInterpolatedP0** implements abstract type: Field:R3 → Enum

---

R3 → Enum Field constant in space.

`TYPE` = *<selection: Field:R3 → Enum_TYPE_selection>*

Default: FieldInterpolatedP0

Sub-record selection.

`gmsh_file` = *<input file name>*

Default: *<obligatory>*

Input file with ASCII GMSH file format.

`field_name` = *<String (generic)>*

Default: *<obligatory>*

The values of the Field are read from the $ElementData section with field name given by this key.

---

record: **FieldElementwise** implements abstract type: Field:R3 → Enum

---

R3 → Enum Field constant in space.

`TYPE` = *<selection: Field:R3 → Enum_TYPE_selection>*

Default: FieldElementwise

Sub-record selection.

`gmsh_file` = *<input file name>*

Default: *<obligatory>*

Input file with ASCII GMSH file format.

`field_name` = *<String (generic)>*

Default: *<obligatory>*

The values of the Field are read from the $ElementData section with field name given by this key.

---

record: **Unsteady_MH** implements abstract type: DarcyFlowMH

Mixed-Hybrid solver for unsteady saturated Darcy flow.

`TYPE` = *<selection: DarcyFlowMH_TYPE_selection>*

Default: Unsteady_MH

Sub-record selection.

`n_schurs` = *<Integer [0, 2]>*

Default: 2

Number of Schur complements to perform when solving MH sytem.

`solver` = *<abstract type: LinSys>*

Default: *<obligatory>*

Linear solver for MH problem.

`output` = *<record: DarcyMHOutput>*

Default: *<obligatory>*

Parameters of output form MH module.

`mortar_method` = *<selection: MH_MortarMethod>*

Default: None

Method for coupling Darcy flow between dimensions.

`input_fields` = *<Array of record: DarcyFlowMH_Data>*

Default: *<obligatory>*

`time` = *<record: TimeGovernor>*

Default: *<obligatory>*

Time governor setting for the unsteady Darcy flow model.

---

record: **Unsteady_LMH** implements abstract type: DarcyFlowMH

Lumped Mixed-Hybrid solver for unsteady saturated Darcy flow.

`TYPE` = *<selection: DarcyFlowMH_TYPE_selection>*

Default: Unsteady_LMH

Sub-record selection.

`n_schurs` = *<Integer [0, 2]>*

Default: 2

Number of Schur complements to perform when solving MH sytem.

`solver` = *<abstract type: LinSys>*

Default: *<obligatory>*

Linear solver for MH problem.

`output` = *<record: DarcyMHOutput>*

Default: *<obligatory>*

Parameters of output form MH module.

`mortar_method` = <*selection:* *MH_MortarMethod*>

Default: None

Method for coupling Darcy flow between dimensions.

`input_fields` = <*Array of record:* *DarcyFlowMH_Data*>

Default: <*obligatory*>

`time` = <*record:* *TimeGovernor*>

Default: <*obligatory*>

Time governor setting for the unsteady Darcy flow model.

---

abstract type: **Transport**

Descendants:

Secondary equation for transport of substances.

`TransportOperatorSplitting`

`SoluteTransport_DG`

`HeatTransfer_DG`

---

record: **TransportOperatorSplitting** implements abstract type: Transport

---

Explicit FVM transport (no diffusion) coupled with reaction and adsorption model (ODE per element) via operator splitting.

`TYPE` = <*selection:* *Transport_TYPE_selection*>

Default: TransportOperatorSplitting

Sub-record selection.

`time` = <*record:* *TimeGovernor*>

Default: <*obligatory*>

Time governor setting for the secondary equation.

`output_stream` = <*record:* *OutputStream*>

Default: <*obligatory*>

Parameters of output stream.

`mass_balance` = <*record:* *MassBalance*>

Default: <*optional*>

Settings for computing mass balance.

`substances` = <*Array of String (generic)*>

Default: <*obligatory*>

Names of transported substances.

reaction_term = <*abstract type: ReactionTerm*>

Default: <*optional*>

Reaction model involved in transport.

input_fields = <*Array of record: TransportOperatorSplitting_Data*>

Default: <*obligatory*>

output_fields = <*Array of selection: ConvectionTransport_Output*>

Default: conc

List of fields to write to output file.

---

record: **MassBalance**

Balance of mass, boundary fluxes and sources for transport of substances.

cumulative = <*Bool*>

Default: false

Compute cumulative balance over time. If true, then balance is calculated at each computational time step, which can slow down the program.

file = <*output file name*>

Default: mass_balance.txt

File name for output of mass balance.

---

abstract type: **ReactionTerm**

Descendants:

Equation for reading information about simple chemical reactions.

LinearReactions

PadeApproximant

Sorption

SorptionMobile

SorptionImmobile

DualPorosity

Isotope

---

record: **LinearReactions** implements abstract type: ReactionTerm

---

Information for a decision about the way to simulate radioactive decay.

TYPE = <*selection: ReactionTerm_TYPE_selection*>

Default: LinearReactions

Sub-record selection.

`decays` = <*Array of record: Substep*>

Default: <*obligatory*>

Description of particular decay chain substeps.

---

record: **Substep**

Equation for reading information about radioactive decays.

`parent` = <*String (generic)*>

Default: <*obligatory*>

Identifier of an isotope.

`half_life` = <*Double* >

Default: <*optional*>

Half life of the parent substance.

`kinetic` = <*Double* >

Default: <*optional*>

Kinetic constants describing first order reactions.

`products` = <*Array of String (generic)*>

Default: <*obligatory*>

Identifies isotopes which decays parental atom to.

`branch_ratios` = <*Array of Double* >

Default: 1.0

Decay chain branching percentage.

---

record: **PadeApproximant** implements abstract type: ReactionTerm

---

Abstract record with an information about pade approximant parameters.

`TYPE` = <*selection: ReactionTerm_TYPE_selection*>

Default: PadeApproximant

Sub-record selection.

`decays` = <*Array of record: Substep*>

Default: <*obligatory*>

Description of particular decay chain substeps.

`nom_pol_deg` = <*Integer* >

Default: 2

Polynomial degree of the nominator of Pade approximant.

`den_pol_deg` = <*Integer* >

Default: 2

Polynomial degree of the nominator of Pade approximant

---

record: **Sorption** implements abstract type: ReactionTerm

Adsorption model in the reaction term of transport.

`TYPE` = <*selection: ReactionTerm_TYPE_selection*>

Default: Sorption

Sub-record selection.

`substances` = <*Array of String (generic)*>

Default: <*obligatory*>

Names of the substances that take part in the adsorption model.

`solvent_density` = <*Double* >

Default: 1.0

Density of the solvent.

`substeps` = <*Integer* >

Default: 1000

Number of equidistant substeps, molar mass and isotherm intersections

`molar_mass` = <*Array of Double* >

Default: <*obligatory*>

Specifies molar masses of all the adsorbing species.

`solubility` = <*Array of Double [0,]*>

Default: <*optional*>

Specifies solubility limits of all the adsorbing species.

`table_limits` = <*Array of Double [0,]*>

Default: <*optional*>

Specifies highest aqueous concentration in interpolation table.

`input_fields` = <*Array of record: Sorption_Data*>

Default: <*obligatory*>

Containes region specific data necessary to construct isotherms.

`reaction_liquid` = <*abstract type: ReactionTerm*>

Default: <*optional*>

Reaction model following the sorption in the liquid.

`reaction_solid` = <*abstract type: ReactionTerm*>

Default: <*optional*>

Reaction model following the sorption in the solid.

`output_fields` = <*Array of selection: Sorption_Output*>

Default: conc_solid

List of fields to write to output stream.

---

record: **Sorption_Data**

Record to set fields of the equation. The fields are set only on the domain specified by one of the keys: 'region', 'rid', 'r_set' and after the time given by the key 'time'. The field setting can be overridden by any Sorption_Data record that comes later in the boundary data array.

`r_set` = *<String (generic)>*

Default: *<optional>*

Name of region set where to set fields.

`region` = *<String (generic)>*

Default: *<optional>*

Label of the region where to set fields.

`rid` = *<Integer [0,]>*

Default: *<optional>*

ID of the region where to set fields.

`time` = *<Double [0,]>*

Default: 0.0

Apply field setting in this record after this time.
These times have to form an increasing sequence.

`rock_density` = *<abstract type: Field:R3 → Real>*

Default: *<optional>*

Rock matrix density.

`sorption_type` = *<abstract type: Field:R3 → Enum[n]>*

Default: *<optional>*

Considered adsorption is described by selected isotherm.

`isotherm_mult` = *<abstract type: Field:R3 → Real[n]>*

Default: *<optional>*

Multiplication parameters (k, omega) in either Langmuir c_s = omega * (alpha*c_a)/(1- alpha*c_a) or in linear c_s = k * c_a isothermal description.

`isotherm_other` = *<abstract type: Field:R3 → Real[n]>*

Default: *<optional>*

Second parameters (alpha, ...) defining isotherm c_s = omega * (alpha*c_a)/(1-alpha*c_a).

`init_conc_solid` = *<abstract type: Field:R3 → Real[n]>*

Default: *<optional>*

Initial solid concentration of substances. Vector, one value for every substance.

---

abstract type: **Field:R3 → Enum[n]** default descendant: FieldConstant

---

Descendants:

Abstract record for all time-space functions.

`FieldConstant`

`FieldFormula`

`FieldPython`

`FieldInterpolatedP0`

`FieldElementwise`

---

record: **FieldConstant** implements abstract type: Field:R3 → Enum[n] constructible from key: value

---

R3 → Enum[n] Field constant in space.

`TYPE` = *<selection: Field:R3 → Enum[n]_TYPE_selection>*

Default: FieldConstant

Sub-record selection.

`value` = *<Array [1,] of selection: AdsorptionType>*

Default: *<obligatory>*

Value of the constant field.
For vector values, you can use scalar value to enter constant vector.
For square NxN-matrix values, you can use:
vector of size N to enter diagonal matrix
vector of size (N+1)*N/2 to enter symmetric matrix (upper triangle, row by row)
scalar to enter multiple of the unit matrix.

---

selection type: **AdsorptionType**

Possible values:

`none` : No adsorption considered.

`linear` : Linear isotherm runs the concentration exchange between liquid and solid.

`langmuir` : Langmuir isotherm runs the concentration exchange between liquid and solid.

`freundlich` : Freundlich isotherm runs the concentration exchange between liquid and solid.

---

record: **FieldFormula** implements abstract type: Field:R3 → Enum[n]

R3 → Enum[n] Field given by runtime interpreted formula.

`TYPE` = *<selection: Field:R3 → Enum[n]_TYPE_selection>*

Default: FieldFormula

Sub-record selection.

`value` = *<Array [1,] of String (generic)>*

Default: *<obligatory>*

String, array of strings, or matrix of strings with formulas for individual entries of scalar, vector, or tensor value respectively.
For vector values, you can use just one string to enter homogeneous vector.
For square NxN-matrix values, you can use:
array of strings of size N to enter diagonal matrix
array of strings of size (N+1)*N/2 to enter symmetric matrix (upper triangle, row by row)
just one string to enter (spatially variable) multiple of the unit matrix.
Formula can contain variables x,y,z,t and usual operators and functions.

---

record: **FieldPython** implements abstract type: Field:R3 → Enum[n]

R3 → Enum[n] Field given by a Python script.

`TYPE` = *<selection: Field:R3 → Enum[n]_TYPE_selection>*

Default: FieldPython

Sub-record selection.

`script_string` = *<String (generic)>*

Default: "Obligatory if 'script_file' is not given."

Python script given as in place string

`script_file` = *<input file name>*

Default: "Obligatory if 'script_striong' is not given."

Python script given as external file

`function` = *<String (generic)>*

Default: *<obligatory>*

Function in the given script that returns tuple containing components of the return type.
For NxM tensor values: tensor(row,col) = tuple(M*row + col).

---

record: **FieldInterpolatedP0** implements abstract type: Field:R3 → Enum[n]

R3 → Enum[n] Field constant in space.

`TYPE` = *<selection: Field:R3 → Enum[n]_TYPE_selection>*

Default: FieldInterpolatedP0

Sub-record selection.

`gmsh_file` = *<input file name>*

Default: *<obligatory>*

Input file with ASCII GMSH file format.

`field_name` = *<String (generic)>*

Default: *<obligatory>*

The values of the Field are read from the $ElementData section with field name given by this key.

---

record: **FieldElementwise** implements abstract type: Field:R3 → Enum[n]

R3 → Enum[n] Field constant in space.

`TYPE` = *<selection: Field:R3 → Enum[n]_TYPE_selection>*

Default: FieldElementwise

Sub-record selection.

`gmsh_file` = *<input file name>*

Default: *<obligatory>*

Input file with ASCII GMSH file format.

`field_name` = *<String (generic)>*

Default: *<obligatory>*

The values of the Field are read from the $ElementData section with field name given by this key.

---

abstract type: **Field:R3 → Real[n]** default descendant: FieldConstant

---

Descendants:

Abstract record for all time-space functions.

`FieldConstant`

`FieldPython`

`FieldFormula`

`FieldElementwise`

`FieldInterpolatedP0`

---

record: **FieldConstant** implements abstract type: Field:R3 → Real[n] constructible from key: value

---

R3 → Real[n] Field constant in space.

`TYPE` = *<selection: Field:R3 → Real[n]_TYPE_selection>*

Default: FieldConstant

Sub-record selection.

`value` = *<Array [1,] of Double >*

Default: *<obligatory>*

Value of the constant field.
For vector values, you can use scalar value to enter constant vector.
For square NxN-matrix values, you can use:
vector of size N to enter diagonal matrix
vector of size (N+1)*N/2 to enter symmetric matrix (upper triangle, row by row)
scalar to enter multiple of the unit matrix.

---

record: **FieldPython** implements abstract type: Field:R3 → Real[n]

R3 → Real[n] Field given by a Python script.

`TYPE` = *<selection: Field:R3 → Real[n]_TYPE_selection>*

Default: FieldPython

Sub-record selection.

`script_string` = *<String (generic)>*

Default: "Obligatory if 'script_file' is not given."

Python script given as in place string

`script_file` = *<input file name>*

Default: "Obligatory if 'script_striong' is not given."

Python script given as external file

`function` = *<String (generic)>*

Default: *<obligatory>*

Function in the given script that returns tuple containing components of the return type.
For NxM tensor values: tensor(row,col) = tuple(M*row + col).

---

record: **FieldFormula** implements abstract type: Field:R3 → Real[n]

R3 → Real[n] Field given by runtime interpreted formula.

`TYPE` = *<selection: Field:R3 → Real[n]_TYPE_selection>*

Default: FieldFormula

Sub-record selection.

`value` = *<Array [1,] of String (generic)>*

Default: *<obligatory>*

String, array of strings, or matrix of strings with formulas for individual entries of scalar, vector, or tensor value respectively.
For vector values, you can use just one string to enter homogeneous vector.

For square NxN-matrix values, you can use:
array of strings of size N to enter diagonal matrix
array of strings of size (N+1)*N/2 to enter symmetric matrix (upper triangle, row by row)
just one string to enter (spatially variable) multiple of the unit matrix.
Formula can contain variables x,y,z,t and usual operators and functions.

---

record: **FieldElementwise** implements abstract type: Field:R3 → Real[n]

R3 → Real[n] Field constant in space.

`TYPE` = <*selection: Field:R3 → Real[n]_TYPE_selection*>

Default: FieldElementwise

Sub-record selection.

`gmsh_file` = <*input file name*>

Default: <*obligatory*>

Input file with ASCII GMSH file format.

`field_name` = <*String (generic)*>

Default: <*obligatory*>

The values of the Field are read from the $ElementData section with field name given by this key.

---

record: **FieldInterpolatedP0** implements abstract type: Field:R3 → Real[n]

R3 → Real[n] Field constant in space.

`TYPE` = <*selection: Field:R3 → Real[n]_TYPE_selection*>

Default: FieldInterpolatedP0

Sub-record selection.

`gmsh_file` = <*input file name*>

Default: <*obligatory*>

Input file with ASCII GMSH file format.

`field_name` = <*String (generic)*>

Default: <*obligatory*>

The values of the Field are read from the $ElementData section with field name given by this key.

---

selection type: **Sorption_Output**

Possible values:

`rock_density` : Output of the field rock_density (Rock matrix density.).

`sorption_type` : Output of the field sorption_type (Considered adsorption is described by selected isotherm.).

`isotherm_mult` : Output of the field isotherm_mult (Multiplication parameters (k, omega) in either Langmuir c_s = omega * (alpha*c_a)/(1- alpha*c_a) or in linear c_s = k * c_a isothermal description.).

`isotherm_other` : Output of the field isotherm_other (Second parameters (alpha, ...) defining isotherm c_s = omega * (alpha*c_a)/(1- alpha*c_a).).

`init_conc_solid` : Output of the field init_conc_solid (Initial solid concentration of substances. Vector, one value for every substance.).

`conc_solid` : Output of the field conc_solid.

---

record: **SorptionMobile** implements abstract type: ReactionTerm

---

Adsorption model in the mobile zone, following the dual porosity model.

`TYPE` = *<selection: ReactionTerm_TYPE_selection>*

Default: SorptionMobile

Sub-record selection.

`substances` = *<Array of String (generic)>*

Default: *<obligatory>*

Names of the substances that take part in the adsorption model.

`solvent_density` = *<Double >*

Default: 1.0

Density of the solvent.

`substeps` = *<Integer >*

Default: 1000

Number of equidistant substeps, molar mass and isotherm intersections

`molar_mass` = *<Array of Double >*

Default: *<obligatory>*

Specifies molar masses of all the adsorbing species.

`solubility` = *<Array of Double [0,]>*

Default: *<optional>*

Specifies solubility limits of all the adsorbing species.

`table_limits` = *<Array of Double [0,]>*

Default: *<optional>*

Specifies highest aqueous concentration in interpolation table.

`input_fields` = *<Array of record: Sorption_Data>*

Default: *<obligatory>*

Containes region specific data necessary to construct isotherms.

`reaction_liquid` = <*abstract type: ReactionTerm*>

Default: <*optional*>

Reaction model following the sorption in the liquid.

`reaction_solid` = <*abstract type: ReactionTerm*>

Default: <*optional*>

Reaction model following the sorption in the solid.

`output_fields` = <*Array of selection: SorptionMobile_Output*>

Default: conc_solid

List of fields to write to output stream.

---

selection type: **SorptionMobile_Output**

Possible values:

`rock_density` : Output of the field rock_density (Rock matrix density.).

`sorption_type` : Output of the field sorption_type (Considered adsorption is described by selected isotherm.).

`isotherm_mult` : Output of the field isotherm_mult (Multiplication parameters (k, omega) in either Langmuir c_s = omega * (alpha*c_a)/(1- alpha*c_a) or in linear c_s = k * c_a isothermal description.).

`isotherm_other` : Output of the field isotherm_other (Second parameters (alpha, ...) defining isotherm c_s = omega * (alpha*c_a)/(1- alpha*c_a).).

`init_conc_solid` : Output of the field init_conc_solid (Initial solid concentration of substances. Vector, one value for every substance.).

`conc_solid` : Output of the field conc_solid.

---

record: **SorptionImmobile** implements abstract type: ReactionTerm

---

Adsorption model in the immobile zone, following the dual porosity model.

`TYPE` = <*selection: ReactionTerm_TYPE_selection*>

Default: SorptionImmobile

Sub-record selection.

`substances` = <*Array of String (generic)*>

Default: <*obligatory*>

Names of the substances that take part in the adsorption model.

`solvent_density` = <*Double* >

Default: 1.0

Density of the solvent.

`substeps` = <*Integer* >

Default: 1000

Number of equidistant substeps, molar mass and isotherm intersections

`molar_mass` = <*Array of Double* >

Default: <*obligatory*>

Specifies molar masses of all the adsorbing species.

`solubility` = <*Array of Double [0,]*>

Default: <*optional*>

Specifies solubility limits of all the adsorbing species.

`table_limits` = <*Array of Double [0,]*>

Default: <*optional*>

Specifies highest aqueous concentration in interpolation table.

`input_fields` = <*Array of record: Sorption_Data*>

Default: <*obligatory*>

Containes region specific data necessary to construct isotherms.

`reaction_liquid` = <*abstract type: ReactionTerm*>

Default: <*optional*>

Reaction model following the sorption in the liquid.

`reaction_solid` = <*abstract type: ReactionTerm*>

Default: <*optional*>

Reaction model following the sorption in the solid.

`output_fields` = <*Array of selection: SorptionImmobile_Output*>

Default: conc_immobile_solid

List of fields to write to output stream.

---

selection type: **SorptionImmobile_Output**

Possible values:

`rock_density` : Output of the field rock_density (Rock matrix density.).

`sorption_type` : Output of the field sorption_type (Considered adsorption is described by selected isotherm.).

`isotherm_mult` : Output of the field isotherm_mult (Multiplication parameters (k, omega) in either Langmuir c_s = omega * (alpha*c_a)/(1- alpha*c_a) or in linear c_s = k * c_a isothermal description.).

`isotherm_other` : Output of the field isotherm_other (Second parameters (alpha, ...) defining isotherm c_s = omega * (alpha*c_a)/(1- alpha*c_a).).

`init_conc_solid` : Output of the field init_conc_solid (Initial solid concentration of

substances. Vector, one value for every substance.).

`conc_immobile_solid` : Output of the field conc_immobile_solid.

---

record: **DualPorosity** implements abstract type: ReactionTerm

---

Dual porosity model in transport problems. Provides computing the concentration of substances in mobile and immobile zone.

`TYPE` = *<selection: ReactionTerm_TYPE_selection>*

Default: DualPorosity

Sub-record selection.

`input_fields` = *<Array of record: DualPorosity_Data>*

Default: *<obligatory>*

Containes region specific data necessary to construct dual porosity model.

`scheme_tolerance` = *<Double [0,]>*

Default: 1e-3

Tolerance according to which the explicit Euler scheme is used or not.Set 0.0 to use analytic formula only (can be slower).

`reaction_mobile` = *<abstract type: ReactionTerm>*

Default: *<optional>*

Reaction model in mobile zone.

`reaction_immobile` = *<abstract type: ReactionTerm>*

Default: *<optional>*

Reaction model in immobile zone.

`output_fields` = *<Array of selection: DualPorosity_Output>*

Default: conc_immobile

List of fields to write to output stream.

---

record: **DualPorosity_Data**

Record to set fields of the equation. The fields are set only on the domain specified by one of the keys: 'region', 'rid', 'r_set' and after the time given by the key 'time'. The field setting can be overridden by any DualPorosity_Data record that comes later in the boundary data array.

`r_set` = *<String (generic)>*

Default: *<optional>*

Name of region set where to set fields.

`region` = *<String (generic)>*

Default: *<optional>*

Label of the region where to set fields.

`rid` = <*Integer [0,]*>

Default: <*optional*>

ID of the region where to set fields.

`time` = <*Double [0,]*>

Default: 0.0

Apply field setting in this record after this time.
These times have to form an increasing sequence.

`diffusion_rate_immobile` = <*abstract type: Field:R3 → Real[n]*>

Default: <*optional*>

Diffusion coefficient of non-equilibrium linear exchange between mobile and immobile zone.

`porosity_immobile` = <*abstract type: Field:R3 → Real*>

Default: <*optional*>

Porosity of the immobile zone.

`init_conc_immobile` = <*abstract type: Field:R3 → Real[n]*>

Default: <*optional*>

Initial concentration of substances in the immobile zone.

---

selection type: **DualPorosity_Output**

Possible values:

`diffusion_rate_immobile` : Output of the field diffusion_rate_immobile (Diffusion coefficient of non-equilibrium linear exchange between mobile and immobile zone.).

`porosity_immobile` : Output of the field porosity_immobile (Porosity of the immobile zone.).

`init_conc_immobile` : Output of the field init_conc_immobile (Initial concentration of substances in the immobile zone.).

`conc_immobile` : Output of the field conc_immobile.

---

record: **Isotope** implements abstract type: ReactionTerm

Definition of information about a single isotope.

`TYPE` = <*selection: ReactionTerm_TYPE_selection*>

Default: Isotope

Sub-record selection.

`identifier` = <*Integer >*

Default: <*obligatory*>

Identifier of the isotope.

`half_life` = <*Double* >

Default: <*obligatory*>

Half life parameter.

---

record: **TransportOperatorSplitting_Data**

Record to set fields of the equation. The fields are set only on the domain specified by one of the keys: 'region', 'rid', 'r_set' and after the time given by the key 'time'. The field setting can be overridden by any TransportOperatorSplitting_Data record that comes later in the boundary data array.

`r_set` = <*String (generic)*>

Default: <*optional*>

Name of region set where to set fields.

`region` = <*String (generic)*>

Default: <*optional*>

Label of the region where to set fields.

`rid` = <*Integer [0,]*>

Default: <*optional*>

ID of the region where to set fields.

`time` = <*Double [0,]*>

Default: 0.0

Apply field setting in this record after this time.
These times have to form an increasing sequence.

`porosity` = <*abstract type: Field:R3 → Real*>

Default: <*optional*>

Mobile porosity

`sources_density` = <*abstract type: Field:R3 → Real[n]*>

Default: <*optional*>

Density of concentration sources.

`sources_sigma` = <*abstract type: Field:R3 → Real[n]*>

Default: <*optional*>

Concentration flux.

`sources_conc` = <*abstract type: Field:R3 → Real[n]*>

Default: <*optional*>

Concentration sources threshold.

`bc_conc` = <*abstract type: Field:R3 → Real[n]*>

Default: <*optional*>

Boundary conditions for concentrations.

`init_conc` = <*abstract type: Field:R3 → Real[n]*>

Default: <*optional*>

Initial concentrations.

`transport_old_bcd_file` = <*input file name*>

Default: <*optional*>

File with mesh dependent boundary conditions (obsolete).

---

selection type: **ConvectionTransport_Output**

Possible values:

`porosity` : Output of the field porosity (Mobile porosity).

`sources_density` : Output of the field sources_density (Density of concentration sources.).

`sources_sigma` : Output of the field sources_sigma (Concentration flux.).

`sources_conc` : Output of the field sources_conc (Concentration sources threshold.).

`init_conc` : Output of the field init_conc (Initial concentrations.).

`conc` : Output of the field conc.

---

record: **SoluteTransport_DG** implements abstract type: Transport

---

DG solver for solute transport.

`TYPE` = <*selection: Transport_TYPE_selection*>

Default: SoluteTransport_DG

Sub-record selection.

`time` = <*record: TimeGovernor*>

Default: <*obligatory*>

Time governor setting for the secondary equation.

`output_stream` = <*record: OutputStream*>

Default: <*obligatory*>

Parameters of output stream.

`mass_balance` = <*record: MassBalance*>

Default: <*optional*>

Settings for computing mass balance.

`substances` = <*Array of String (generic)*>

Default: <*obligatory*>

Names of transported substances.

`solver` = <*record: Petsc*>

Default: <*obligatory*>

Linear solver for MH problem.

`input_fields` = <*Array of record: SoluteTransport_DG_Data*>

Default: <*obligatory*>

`dg_variant` = <*selection: DG_variant*>

Default: non-symmetric

Variant of interior penalty discontinuous Galerkin method.

`dg_order` = <*Integer [0, 3]*>

Default: 1

Polynomial order for finite element in DG method (order 0 is suitable if there is no diffusion/dispersion).

`output_fields` = <*Array of selection: SoluteTransport_DG_Output*>

Default: conc

List of fields to write to output file.

---

record: **SoluteTransport_DG_Data**

Record to set fields of the equation. The fields are set only on the domain specified by one of the keys: 'region', 'rid', 'r_set' and after the time given by the key 'time'. The field setting can be overridden by any SoluteTransport_DG_Data record that comes later in the boundary data array.

`r_set` = <*String (generic)*>

Default: <*optional*>

Name of region set where to set fields.

`region` = <*String (generic)*>

Default: <*optional*>

Label of the region where to set fields.

`rid` = <*Integer [0,]*>

Default: <*optional*>

ID of the region where to set fields.

`time` = <*Double [0,]*>

Default: 0.0

Apply field setting in this record after this time.
These times have to form an increasing sequence.

`porosity` = <*abstract type: Field:R3 → Real*>

Default: <*optional*>

Mobile porosity

`sources_density` = <*abstract type: Field:R3 → Real[n]*>

Default: <*optional*>

Density of concentration sources.

`sources_sigma` = <*abstract type: Field:R3 → Real[n]*>

Default: <*optional*>

Concentration flux.

`sources_conc` = <*abstract type: Field:R3 → Real[n]*>

Default: <*optional*>

Concentration sources threshold.

`bc_conc` = <*abstract type: Field:R3 → Real[n]*>

Default: <*optional*>

Dirichlet boundary condition (for each substance).

`init_conc` = <*abstract type: Field:R3 → Real[n]*>

Default: <*optional*>

Initial concentrations.

`disp_l` = <*abstract type: Field:R3 → Real[n]*>

Default: <*optional*>

Longitudal dispersivity (for each substance).

`disp_t` = <*abstract type: Field:R3 → Real[n]*>

Default: <*optional*>

Transversal dispersivity (for each substance).

`diff_m` = <*abstract type: Field:R3 → Real[n]*>

Default: <*optional*>

Molecular diffusivity (for each substance).

`fracture_sigma` = <*abstract type: Field:R3 → Real[n]*>

Default: <*optional*>

Coefficient of diffusive transfer through fractures (for each substance).

`dg_penalty` = <*abstract type: Field:R3 → Real[n]*>

Default: <*optional*>

Penalty parameter influencing the discontinuity of the solution (for each substance). Its default value 1 is sufficient in most cases. Higher value diminishes the inter-element jumps.

`bc_type` = <*abstract type: Field:R3 → Enum[n]*>

Default: <*optional*>

Boundary condition type, possible values: inflow, dirichlet, neumann, robin.

`bc_flux` = <*abstract type: Field:R3 → Real[n]*>

Default: <*optional*>

Flux in Neumann boundary condition.

`bc_robin_sigma` = <*abstract type: Field:R3 → Real[n]*>

Default: <*optional*>

Conductivity coefficient in Robin boundary condition.

---

selection type: **DG_variant**

Type of penalty term.

Possible values:

`non-symmetric` : non-symmetric weighted interior penalty DG method

`incomplete` : incomplete weighted interior penalty DG method

`symmetric` : symmetric weighted interior penalty DG method

---

selection type: **SoluteTransport_DG_Output**

Output record for DG solver for solute transport.

Possible values:

`porosity` : Output of the field porosity (Mobile porosity).

`sources_density` : Output of the field sources_density (Density of concentration sources.).

`sources_sigma` : Output of the field sources_sigma (Concentration flux.).

`sources_conc` : Output of the field sources_conc (Concentration sources threshold.).

`init_conc` : Output of the field init_conc (Initial concentrations.).

`disp_l` : Output of the field disp_l (Longitudal dispersivity (for each substance).).

`disp_t` : Output of the field disp_t (Transversal dispersivity (for each substance).).

`diff_m` : Output of the field diff_m (Molecular diffusivity (for each substance).).

`conc` : Output of the field conc.

`fracture_sigma` : Output of the field fracture_sigma (Coefficient of diffusive transfer through fractures (for each substance).).

`dg_penalty` : Output of the field dg_penalty (Penalty parameter influencing the discontinuity of the solution (for each substance). Its default value 1 is sufficient in most cases. Higher value diminishes the inter-element jumps.).

---

record: **HeatTransfer_DG** implements abstract type: Transport

---

DG solver for heat transfer.

`TYPE` = <*selection: Transport_TYPE_selection*>

Default: HeatTransfer_DG

Sub-record selection.

`time` = <*record: TimeGovernor*>

Default: <*obligatory*>

Time governor setting for the secondary equation.

`output_stream` = <*record: OutputStream*>

Default: <*obligatory*>

Parameters of output stream.

`mass_balance` = <*record: MassBalance*>

Default: <*optional*>

Settings for computing mass balance.

`solver` = <*record: Petsc*>

Default: <*obligatory*>

Linear solver for MH problem.

`input_fields` = <*Array of record: HeatTransfer_DG_Data*>

Default: <*obligatory*>

`dg_variant` = <*selection: DG_variant*>

Default: non-symmetric

Variant of interior penalty discontinuous Galerkin method.

`dg_order` = <*Integer [0, 3]*>

Default: 1

Polynomial order for finite element in DG method (order 0 is suitable if there is no diffusion/dispersion).

`output_fields` = <*Array of selection: HeatTransfer_DG_Output*>

Default: temperature

List of fields to write to output file.

---

record: **HeatTransfer_DG_Data**

Record to set fields of the equation. The fields are set only on the domain specified by one of the keys: 'region', 'rid', 'r_set' and after the time given by the key 'time'. The field setting can be overridden by any HeatTransfer_DG_Data record that comes later in the boundary data array.

`r_set` = <*String (generic)*>

Default: <*optional*>

Name of region set where to set fields.

`region` = <*String (generic)*>

Default: <*optional*>

Label of the region where to set fields.

`rid` = <*Integer [0,]*>

Default: <*optional*>

ID of the region where to set fields.

`time` = <*Double [0,]*>

Default: 0.0

Apply field setting in this record after this time.
These times have to form an increasing sequence.

`bc_temperature` = <*abstract type: Field:R3 → Real*>

Default: <*optional*>

Boundary value of temperature.

`init_temperature` = <*abstract type: Field:R3 → Real*>

Default: <*optional*>

Initial temperature.

`porosity` = <*abstract type: Field:R3 → Real*>

Default: <*optional*>

Porosity.

`fluid_density` = <*abstract type: Field:R3 → Real*>

Default: <*optional*>

Density of fluid.

`fluid_heat_capacity` = <*abstract type: Field:R3 → Real*>

Default: <*optional*>

Heat capacity of fluid.

`fluid_heat_conductivity` = <*abstract type: Field:R3 → Real*>

Default: <*optional*>

Heat conductivity of fluid.

`solid_density` = <*abstract type: Field:R3 → Real*>

Default: <*optional*>

Density of solid (rock).

`solid_heat_capacity` = <*abstract type: Field:R3 → Real*>

Default: <*optional*>

Heat capacity of solid (rock).

`solid_heat_conductivity` = <*abstract type: Field:R3 → Real*>

Default: <*optional*>

Heat conductivity of solid (rock).

`disp_l` = <*abstract type: Field:R3 → Real*>

Default: <*optional*>

Longitudal heat dispersivity in fluid.

`disp_t` = <*abstract type: Field:R3 → Real*>

Default: <*optional*>

Transversal heat dispersivity in fluid.

`fluid_thermal_source` = <*abstract type: Field:R3 → Real*>

Default: <*optional*>

Thermal source density in fluid.

`solid_thermal_source` = <*abstract type: Field:R3 → Real*>

Default: <*optional*>

Thermal source density in solid.

`fluid_heat_exchange_rate` = <*abstract type: Field:R3 → Real*>

Default: <*optional*>

Heat exchange rate in fluid.

`solid_heat_exchange_rate` = <*abstract type: Field:R3 → Real*>

Default: <*optional*>

Heat exchange rate of source in solid.

`fluid_ref_temperature` = <*abstract type: Field:R3 → Real*>

Default: <*optional*>

Reference temperature of source in fluid.

`solid_ref_temperature` = <*abstract type: Field:R3 → Real*>

Default: <*optional*>

Reference temperature in solid.

`fracture_sigma` = <*abstract type: Field:R3 → Real[n]*>

Default: <*optional*>

Coefficient of diffusive transfer through fractures (for each substance).

`dg_penalty` = <*abstract type: Field:R3 → Real[n]*>

Default: <*optional*>

Penalty parameter influencing the discontinuity of the solution (for each substance). Its default value 1 is sufficient in most cases. Higher value diminishes the inter-element jumps.

`bc_type` = <*abstract type: Field:R3 → Enum[n]*>

Default: <*optional*>

Boundary condition type, possible values: inflow, dirichlet, neumann, robin.

`bc_flux` = <*abstract type: Field:R3 → Real[n]*>

Default: <*optional*>

Flux in Neumann boundary condition.

`bc_robin_sigma` = <*abstract type: Field:R3 → Real[n]*>

Default: <*optional*>

Conductivity coefficient in Robin boundary condition.

---

selection type: **HeatTransfer_DG_Output**

Selection for output fields of DG solver for heat transfer.

Possible values:

`init_temperature` : Output of the field init_temperature (Initial temperature.).

`porosity` : Output of the field porosity (Porosity.).

`fluid_density` : Output of the field fluid_density (Density of fluid.).

`fluid_heat_capacity` : Output of the field fluid_heat_capacity (Heat capacity of fluid.).

`fluid_heat_conductivity` : Output of the field fluid_heat_conductivity (Heat conductivity of fluid.).

`solid_density` : Output of the field solid_density (Density of solid (rock).).

`solid_heat_capacity` : Output of the field solid_heat_capacity (Heat capacity of solid (rock).).

`solid_heat_conductivity` : Output of the field solid_heat_conductivity (Heat conductivity of solid (rock).).

`disp_l` : Output of the field disp_l (Longitudal heat dispersivity in fluid.).

`disp_t` : Output of the field disp_t (Transversal heat dispersivity in fluid.).

`fluid_thermal_source` : Output of the field fluid_thermal_source (Thermal source density in fluid.).

`solid_thermal_source` : Output of the field solid_thermal_source (Thermal source density in solid.).

`fluid_heat_exchange_rate` : Output of the field fluid_heat_exchange_rate (Heat exchange rate in fluid.).

`solid_heat_exchange_rate` : Output of the field solid_heat_exchange_rate (Heat exchange rate of source in solid.).

`fluid_ref_temperature` : Output of the field fluid_ref_temperature (Reference temperature of source in fluid.).

`solid_ref_temperature` : Output of the field solid_ref_temperature (Reference temperature in solid.).

`temperature` : Output of the field temperature.

`fracture_sigma` : Output of the field fracture_sigma (Coefficient of diffusive transfer through fractures (for each substance).).

`dg_penalty` : Output of the field dg_penalty (Penalty parameter influencing the discontinuity of the solution (for each substance). Its default value 1 is sufficient in most cases. Higher value diminishes the inter-element jumps.).

Bibliography

[1] M. Číslerová and T. Vogel. *Transportní procesy*. ČVUT, 1998.

[2] G. De Marsily. *Quantitative hydrogeology: Groundwater hydrology for engineers*. Academic Press, New York, 1986.

[3] P. A. Domenico and F. W. Schwartz. *Physical and chemical hydrogeology*, volume 824. Wiley New York, 1990.

[4] V. Martin, J. Jaffré, and J. E. Roberts. Modeling fractures and barriers as interfaces for flow in porous media. *SIAM Journal on Scientific Computing*, 26(5):1667, 2005. ISSN 10648275. doi: 10.1137/S1064827503429363. URL http://link.aip.org/link/SJOCE3/v26/i5/p1667/s1&Agg=doi.

[5] R. Millington and J. Quirk. Permeability of porous solids. *Transactions of the Faraday Society*, 57:1200–1207, 1961.