Technical university of Liberec

Faculty of mechatronics, informatics
and interdisciplinary studies

Flow123d

version 1.8.JHy_python_update

Documentation of file formats
and brief user manual.

Liberec, 2015

Authors:

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

Acknowledgement

This work was supported by the Ministry of Industry and Trade of the Czech Republic under the project no. FR-TI3/579.

record: **IT::Root**

Root record of JSON input for Flow123d.

`Root::problem` = ⟨*abstract type: IT::Problem* ⟩
Default: ⟨*obligatory* ⟩
Simulation problem to be solved.

`Root::pause_after_run` = ⟨*Bool* ⟩
Default: false
If true, the program will wait for key press before it terminates.

abstract type: **IT::Problem**

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

`IT::SequentialCoupling`

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

Record with data for a general sequential coupling.

`SequentialCoupling::TYPE` = ⟨*selection: Problem_TYPE_selection* ⟩
Default: SequentialCoupling
Sub-record selection.

`SequentialCoupling::description` = ⟨*String (generic)* ⟩
Default: ⟨*optional* ⟩
Short description of the solved problem.
Is displayed in the main log, and possibly in other text output files.

`SequentialCoupling::mesh` = ⟨*record: IT::Mesh* ⟩
Default: ⟨*obligatory* ⟩
Computational mesh common to all equations.

`SequentialCoupling::time` = ⟨*record: IT::TimeGovernor* ⟩
Default: ⟨*optional* ⟩
Simulation time frame and time step.

`SequentialCoupling::primary_equation` = ⟨*abstract type: IT::DarcyFlowMH* ⟩
Default: ⟨*obligatory* ⟩
Primary equation, have all data given.

`SequentialCoupling::secondary_equation` = ⟨*abstract type: IT::Transport* ⟩
Default: ⟨*optional* ⟩
The equation that depends (the velocity field) on the result of the primary equation.

record: **IT::Mesh**

Record with mesh related data.

`Mesh::mesh_file` = ⟨*input file name* ⟩
Default: ⟨*obligatory* ⟩
Input file with mesh description.

`Mesh::regions` = ⟨*Array of Record: IT::Region* ⟩
Default: ⟨*optional* ⟩
List of additional region definitions not contained in the mesh.

`Mesh::sets` = ⟨*Array of Record: IT::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)

`Mesh::partitioning` = ⟨*record: IT::Partition* ⟩

Default: ⟨*any_neighboring* ⟩

Parameters of mesh partitioning algorithms.

---

record: **IT::Region**

Definition of region of elements.

`Region::name` = ⟨*String (generic)* ⟩

Default: ⟨*obligatory* ⟩

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

`Region::id` = ⟨*Integer[0,]* ⟩

Default: ⟨*obligatory* ⟩

The ID of the region to which you assign label.

`Region::element_list` = ⟨*Array of Integer [0,]* ⟩

Default: ⟨*optional* ⟩

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

---

record: **IT::RegionSet**

Definition of one region set.

`RegionSet::name` = ⟨*String (generic)* ⟩

Default: ⟨*obligatory* ⟩

Unique name of the region set.

`RegionSet::region_ids` = ⟨*Array of Integer [0,]* ⟩

Default: ⟨*optional* ⟩

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

`RegionSet::region_labels` = ⟨*Array of String (generic)* ⟩

Default: ⟨*optional* ⟩

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

`RegionSet::union` = ⟨*Array [2, 2] of String (generic)* ⟩

Default: ⟨*optional* ⟩

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

`RegionSet::intersection` = ⟨*Array [2, 2] of String (generic)* ⟩

Default: ⟨*optional* ⟩

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

`RegionSet::difference` = ⟨*Array [2, 2] of String (generic)* ⟩

Default: ⟨*optional* ⟩

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

---

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

---

Setting for various types of mesh partitioning.

`Partition::tool` = ⟨*selection: IT::PartTool* ⟩
Default: METIS
Software package used for partitioning. See corresponding selection.

`Partition::graph_type` = ⟨*selection: IT::GraphType* ⟩
Default: any_neighboring
Algorithm for generating graph and its weights from a multidimensional mesh.

---

selection type: **IT::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: **IT::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: **IT::TimeGovernor** constructible from key: TimeGovernor::max_dt

---

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

`TimeGovernor::start_time` = ⟨*Double* ⟩
Default: ⟨*0.0* ⟩
Start time of the simulation.

`TimeGovernor::end_time` = ⟨*Double* ⟩
Default: Infinite end time.
End time of the simulation.

`TimeGovernor::init_dt` = ⟨*Double[0,]* ⟩
Default: ⟨*0.0* ⟩
Initial guess for the time step.
Only useful for equations that use adaptive time stepping.If set to 0.0, the time step is determined in fully autonomous way if the equation supports it.

`TimeGovernor::min_dt` = ⟨*Double[0,]* ⟩
Default: Machine precision.
Soft lower limit for the time step. Equation using adaptive time stepping can notsuggest smaller time step, but actual time step could be smaller in order to match prescribed input or output times.

`TimeGovernor::max_dt` = ⟨*Double[0,]* ⟩
Default: Whole time of the simulation if specified, infinity else.
Hard upper limit for the time step. Actual length of the time step is also limitedby input and output times.

abstract type: **IT::DarcyFlowMH**

Descendants:

Mixed-Hybrid solver for saturated Darcy flow.

`IT::Steady_MH`

`IT::Unsteady_MH`

`IT::Unsteady_LMH`

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

Mixed-Hybrid solver for STEADY saturated Darcy flow.

`Steady_MH::TYPE` = ⟨*selection: DarcyFlowMH_TYPE_selection* ⟩
Default: Steady_MH
Sub-record selection.

`Steady_MH::n_schurs` = ⟨*Integer[0, 2]* ⟩
Default: ⟨*2* ⟩
Number of Schur complements to perform when solving MH sytem.

`Steady_MH::solver` = ⟨*abstract type: IT::LinSys* ⟩
Default: ⟨*obligatory* ⟩
Linear solver for MH problem.

`Steady_MH::output` = ⟨*record: IT::DarcyMHOutput* ⟩
Default: ⟨*obligatory* ⟩
Parameters of output form MH module.

`Steady_MH::mortar_method` = ⟨*selection: IT::MH_MortarMethod* ⟩
Default: None
Method for coupling Darcy flow between dimensions.

`Steady_MH::balance` = ⟨*record: IT::Balance* ⟩
Default: ⟨*obligatory* ⟩
Settings for computing mass balance.

`Steady_MH::input_fields` = ⟨*Array of Record: IT::DarcyFlowMH_Data* ⟩
Default: ⟨*obligatory* ⟩

abstract type: **IT::LinSys**

Descendants:

Linear solver setting.

`IT::Petsc`

`IT::Bddc`

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

Solver setting.

`Petsc::TYPE` = ⟨*selection: LinSys_TYPE_selection* ⟩
Default: Petsc
Sub-record selection.

`Petsc::r_tol` = ⟨*Double[0, 1]* ⟩
Default: ⟨*1.0e-7* ⟩
Relative residual tolerance (to initial error).

`Petsc::max_it` = ⟨*Integer[0,]* ⟩

Default: ⟨*10000*⟩
Maximum number of outer iterations of the linear solver.

`Petsc::a_tol` = ⟨*Double[0,]*⟩
Default: ⟨*1.0e-9*⟩
Absolute residual tolerance.

`Petsc::options` = ⟨*String (generic)*⟩
Default: ⟨*value at declaration*⟩
Options passed to PETSC before creating KSP instead of default setting.

---

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

Solver setting.

`Bddc::TYPE` = ⟨*selection: LinSys_TYPE_selection*⟩
Default: Bddc
Sub-record selection.

`Bddc::r_tol` = ⟨*Double[0, 1]*⟩
Default: ⟨*1.0e-7*⟩
Relative residual tolerance (to initial error).

`Bddc::max_it` = ⟨*Integer[0,]*⟩
Default: ⟨*10000*⟩
Maximum number of outer iterations of the linear solver.

`Bddc::max_nondecr_it` = ⟨*Integer[0,]*⟩
Default: ⟨*30*⟩
Maximum number of iterations of the linear solver with non-decreasing residual.

`Bddc::number_of_levels` = ⟨*Integer[0,]*⟩
Default: ⟨*2*⟩
Number of levels in the multilevel method (=2 for the standard BDDC).

`Bddc::use_adaptive_bddc` = ⟨*Bool*⟩
Default: false
Use adaptive selection of constraints in BDDCML.

`Bddc::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: **IT::DarcyMHOutput**

Parameters of MH output.

`DarcyMHOutput::output_stream` = ⟨*record: IT::OutputStream*⟩
Default: ⟨*obligatory*⟩
Parameters of output stream.

`DarcyMHOutput::output_fields` = ⟨*Array of Selection: IT::DarcyMHOutput_Selection*⟩
Default: ⟨*obligatory*⟩
List of fields to write to output file.

`DarcyMHOutput::compute_errors` = ⟨*Bool*⟩
Default: false

SPECIAL PURPOSE. Computing errors pro non-compatible coupling.

`DarcyMHOutput::raw_flow_output` = ⟨*output file name* ⟩

Default: ⟨*optional* ⟩

Output file with raw data form MH module.

---

record: **IT::OutputStream**

Parameters of output.

`OutputStream::file` = ⟨*output file name* ⟩

Default: ⟨*obligatory* ⟩

File path to the connected output file.

`OutputStream::format` = ⟨*abstract type: IT::OutputTime* ⟩

Default: ⟨*optional* ⟩

Format of output stream and possible parameters.

`OutputStream::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.

`OutputStream::time_list` = ⟨*Array of Double* ⟩

Default: List containing the initial time of the equation. You can prescribe an empty list to override this behavior.

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

`OutputStream::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: **IT::OutputTime**

Descendants:

Format of output stream and possible parameters.

`IT::vtk`

`IT::gmsh`

---

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

Parameters of vtk output format.

`vtk::TYPE` = ⟨*selection: OutputTime_TYPE_selection* ⟩

Default: vtk

Sub-record selection.

`vtk::variant` = ⟨*selection: IT::VTK variant (ascii or binary)* ⟩

Default: ascii

Variant of output stream file format.

`vtk::parallel` = ⟨*Bool* ⟩

Default: false

Parallel or serial version of file format.

`vtk::compression` = ⟨*selection: IT::Type of compression of VTK file format* ⟩

Default: none
Compression used in output stream file format.

selection type: **IT::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: **IT::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: **IT::gmsh** implements abstract type: IT::OutputTime

Parameters of gmsh output format.

`gmsh::TYPE` = ⟨*selection: OutputTime_TYPE_selection* ⟩
Default: gmsh
Sub-record selection.

selection type: **IT::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 $[m^{3-d}]$ (Complement dimension parameter (cross section for 1D, thickness for 2D).).

`conductivity` : Output of the field conductivity $[ms^{-1}]$ (Isotropic conductivity scalar.).

`sigma` : Output of the field sigma $[-]$ (Transition coefficient between dimensions.).

`water_source_density` : Output of the field water_source_density $[s^{-1}]$ (Water source density.).

`init_pressure` : Output of the field init_pressure $[m]$ (Initial condition as pressure).

`storativity` : Output of the field storativity $[m^{-1}]$ (Storativity.).

`pressure_p0` : Output of the field pressure_p0 $[m]$.

`pressure_p1` : Output of the field pressure_p1 $[m]$.

`piezo_head_p0` : Output of the field piezo_head_p0 $[m]$.

`velocity_p0` : Output of the field velocity_p0 $[ms^{-1}]$.

`subdomain` : Output of the field subdomain $[-]$.

`region_id` : Output of the field region_id $[-]$.

`pressure_diff` : Output of the field pressure_diff $[m]$.

`velocity_diff` : Output of the field velocity_diff $[ms^{-1}]$.

`div_diff` : Output of the field div_diff $[s^{-1}]$.

selection type: **IT::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: **IT::Balance** constructible from key: Balance::balance_on

---

Balance of a conservative quantity, boundary fluxes and sources.

`Balance::balance_on` = ⟨*Bool*⟩
Default: true
Balance is computed if the value is true.

`Balance::format` = ⟨*selection: IT::Balance_output_format*⟩
Default: txt
Format of output file.

`Balance::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.

`Balance::file` = ⟨*output file name*⟩
Default: FileName balance.*
File name for output of balance.

---

selection type: **IT::Balance_output_format**

Format of output file for balance.
Possible values:

`legacy` : Legacy format used by previous program versions.

`txt` : Excel format with tab delimiter.

`gnuplot` : Format compatible with GnuPlot datafile with fixed column width.

---

record: **IT::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.

`DarcyFlowMH_Data::r_set` = ⟨*String (generic)*⟩
Default: ⟨*optional*⟩
Name of region set where to set fields.

`DarcyFlowMH_Data::region` = ⟨*String (generic)*⟩
Default: ⟨*optional*⟩
Label of the region where to set fields.

`DarcyFlowMH_Data::rid` = ⟨*Integer[0,]*⟩
Default: ⟨*optional*⟩
ID of the region where to set fields.

`DarcyFlowMH_Data::time` = ⟨*Double[0,]*⟩
Default: ⟨*0.0*⟩
Apply field setting in this record after this time.
These times have to form an increasing sequence.

`DarcyFlowMH_Data::anisotropy` = ⟨*abstract type: IT::Field:R3 → Real[3,3]* ⟩
Default: ⟨*optional* ⟩
Anisotropy of the conductivity tensor. [−]

`DarcyFlowMH_Data::cross_section` = ⟨*abstract type: IT::Field:R3 → Real* ⟩
Default: ⟨*optional* ⟩
Complement dimension parameter (cross section for 1D, thickness for 2D). $[m^{3-d}]$

`DarcyFlowMH_Data::conductivity` = ⟨*abstract type: IT::Field:R3 → Real* ⟩
Default: ⟨*optional* ⟩
Isotropic conductivity scalar. $[ms^{-1}]$

`DarcyFlowMH_Data::sigma` = ⟨*abstract type: IT::Field:R3 → Real* ⟩
Default: ⟨*optional* ⟩
Transition coefficient between dimensions. [−]

`DarcyFlowMH_Data::water_source_density` = ⟨*abstract type: IT::Field:R3 → Real* ⟩
Default: ⟨*optional* ⟩
Water source density. $[s^{-1}]$

`DarcyFlowMH_Data::bc_type` = ⟨*abstract type: IT::Field:R3 → Enum* ⟩
Default: ⟨*optional* ⟩
Boundary condition type, possible values: [−]

`DarcyFlowMH_Data::bc_pressure` = ⟨*abstract type: IT::Field:R3 → Real* ⟩
Default: ⟨*optional* ⟩
Dirichlet BC condition value for pressure. $[m]$

`DarcyFlowMH_Data::bc_flux` = ⟨*abstract type: IT::Field:R3 → Real* ⟩
Default: ⟨*optional* ⟩
Flux in Neumman or Robin boundary condition. $[m^{4-d}s^{-1}]$

`DarcyFlowMH_Data::bc_robin_sigma` = ⟨*abstract type: IT::Field:R3 → Real* ⟩
Default: ⟨*optional* ⟩
Conductivity coefficient in Robin boundary condition. $[m^{3-d}s^{-1}]$

`DarcyFlowMH_Data::init_pressure` = ⟨*abstract type: IT::Field:R3 → Real* ⟩
Default: ⟨*optional* ⟩
Initial condition as pressure $[m]$

`DarcyFlowMH_Data::storativity` = ⟨*abstract type: IT::Field:R3 → Real* ⟩
Default: ⟨*optional* ⟩
Storativity. $[m^{-1}]$

`DarcyFlowMH_Data::bc_piezo_head` = ⟨*abstract type: IT::Field:R3 → Real* ⟩
Default: ⟨*optional* ⟩
Boundary condition for pressure as piezometric head.

`DarcyFlowMH_Data::init_piezo_head` = ⟨*abstract type: IT::Field:R3 → Real* ⟩
Default: ⟨*optional* ⟩
Initial condition for pressure as piezometric head.

`DarcyFlowMH_Data::flow_old_bcd_file` = ⟨*input file name* ⟩
Default: ⟨*optional* ⟩
File with mesh dependent boundary conditions (obsolete).

---

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

---

Descendants:

Abstract record for all time-space functions.

IT::FieldConstant

IT::FieldPython

IT::FieldFormula

IT::FieldElementwise

IT::FieldInterpolatedP0

---

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

---

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

`FieldConstant::TYPE` = ⟨*selection: Field:R3 → Real[3,3]_TYPE_selection* ⟩
Default: FieldConstant
Sub-record selection.

`FieldConstant::value` = ⟨*Array [1,] of Array* ⟩
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: **IT::FieldPython** implements abstract type: IT::Field:R3 → Real[3,3]

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

`FieldPython::TYPE` = ⟨*selection: Field:R3 → Real[3,3]_TYPE_selection* ⟩
Default: FieldPython
Sub-record selection.

`FieldPython::script_string` = ⟨*String (generic)* ⟩
Default: ⟨*value at read time* ⟩
Python script given as in place string

`FieldPython::script_file` = ⟨*input file name* ⟩
Default: Obligatory if 'script_striong' is not given.
Python script given as external file

`FieldPython::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: **IT::FieldFormula** implements abstract type: IT::Field:R3 → Real[3,3]

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

`FieldFormula::TYPE` = ⟨*selection: Field:R3 → Real[3,3]_TYPE_selection* ⟩
Default: FieldFormula
Sub-record selection.

`FieldFormula::value` = ⟨*Array [1,] of Array* ⟩
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: **IT::FieldElementwise** implements abstract type: IT::Field:R3 → Real[3,3]

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

`FieldElementwise::TYPE` = ⟨*selection: Field:R3 → Real[3,3]_TYPE_selection* ⟩
Default: FieldElementwise
Sub-record selection.

`FieldElementwise::gmsh_file` = ⟨*input file name* ⟩
Default: ⟨*obligatory* ⟩
Input file with ASCII GMSH file format.

`FieldElementwise::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: **IT::FieldInterpolatedP0** implements abstract type: IT::Field:R3 → Real[3,3]

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

`FieldInterpolatedP0::TYPE` = ⟨*selection: Field:R3 → Real[3,3]_TYPE_selection* ⟩
Default: FieldInterpolatedP0
Sub-record selection.

`FieldInterpolatedP0::gmsh_file` = ⟨*input file name* ⟩
Default: ⟨*obligatory* ⟩
Input file with ASCII GMSH file format.

`FieldInterpolatedP0::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: **IT::Field:R3 → Real** default descendant: IT::FieldConstant

---

Descendants:
Abstract record for all time-space functions.

`IT::FieldConstant`

`IT::FieldPython`

`IT::FieldFormula`

`IT::FieldElementwise`

`IT::FieldInterpolatedP0`

---

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

R3 → Real Field constant in space.

`FieldConstant::TYPE` = ⟨*selection: Field:R3 → Real_TYPE_selection* ⟩
Default: FieldConstant
Sub-record selection.

`FieldConstant::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: **IT::FieldPython** implements abstract type: IT::Field:R3 → Real

R3 → Real Field given by a Python script.

`FieldPython::TYPE` = ⟨*selection: Field:R3 → Real_TYPE_selection* ⟩
Default: FieldPython
Sub-record selection.

`FieldPython::script_string` = ⟨*String (generic)* ⟩
Default: ⟨*value at read time* ⟩
Python script given as in place string

`FieldPython::script_file` = ⟨*input file name* ⟩
Default: Obligatory if 'script_striong' is not given.
Python script given as external file

`FieldPython::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: **IT::FieldFormula** implements abstract type: IT::Field:R3 → Real

R3 → Real Field given by runtime interpreted formula.

`FieldFormula::TYPE` = ⟨*selection: Field:R3 → Real_TYPE_selection* ⟩
Default: FieldFormula
Sub-record selection.

`FieldFormula::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: **IT::FieldElementwise** implements abstract type: IT::Field:R3 → Real

---

R3 → Real Field constant in space.

`FieldElementwise::TYPE` = ⟨*selection: Field:R3 → Real_TYPE_selection* ⟩
Default: FieldElementwise
Sub-record selection.

`FieldElementwise::gmsh_file` = ⟨*input file name* ⟩
Default: ⟨*obligatory* ⟩
Input file with ASCII GMSH file format.

`FieldElementwise::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: **IT::FieldInterpolatedP0** implements abstract type: IT::Field:R3 → Real

---

R3 → Real Field constant in space.

`FieldInterpolatedP0::TYPE` = ⟨*selection: Field:R3 → Real_TYPE_selection* ⟩
Default: FieldInterpolatedP0
Sub-record selection.

`FieldInterpolatedP0::gmsh_file` = ⟨*input file name* ⟩
Default: ⟨*obligatory* ⟩
Input file with ASCII GMSH file format.

`FieldInterpolatedP0::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: **IT::Field:R3 → Enum** default descendant: IT::FieldConstant

---

Descendants:
Abstract record for all time-space functions.

`IT::FieldConstant`

`IT::FieldFormula`

`IT::FieldPython`

`IT::FieldInterpolatedP0`

`IT::FieldElementwise`

---

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

---

R3 → Enum Field constant in space.

`FieldConstant::TYPE` = ⟨*selection: Field:R3 → Enum_TYPE_selection* ⟩
Default: FieldConstant
Sub-record selection.

`FieldConstant::value` = ⟨*selection: IT::DarcyFlow_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: **IT::DarcyFlow_BC_Type**

Possible values:

`none` : Homogeneous Neumann boundary condition. Zero flux

`dirichlet` : Dirichlet boundary condition. Specify the pressure head through the 'bc_pressure' field or the piezometric head through the 'bc_piezo_head' field.

`neumann` : Neumann boundary condition. Prescribe water outflow by the 'bc_flux' field.

`robin` : Robin boundary condition. Water outflow equal to $sigma(h - h^R)$. Specify the transition coefficient by 'bc_sigma' and the reference pressure head or pieaozmetric head through 'bc_pressure' and 'bc_piezo_head' respectively.

---

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

---

R3 → Enum Field given by runtime interpreted formula.

`FieldFormula::TYPE` = ⟨*selection: Field:R3 → Enum_TYPE_selection* ⟩

Default: FieldFormula

Sub-record selection.

`FieldFormula::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: **IT::FieldPython** implements abstract type: IT::Field:R3 → Enum

---

R3 → Enum Field given by a Python script.

`FieldPython::TYPE` = ⟨*selection: Field:R3 → Enum_TYPE_selection* ⟩

Default: FieldPython

Sub-record selection.

`FieldPython::script_string` = ⟨*String (generic)* ⟩

Default: ⟨*value at read time* ⟩

Python script given as in place string

`FieldPython::script_file` = ⟨*input file name* ⟩

Default: Obligatory if 'script_striong' is not given.

Python script given as external file

`FieldPython::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: **IT::FieldInterpolatedP0** implements abstract type: IT::Field:R3 → Enum

---

R3 → Enum Field constant in space.

`FieldInterpolatedP0::TYPE` = ⟨*selection: Field:R3 → Enum_TYPE_selection* ⟩

Default: FieldInterpolatedP0

Sub-record selection.

`FieldInterpolatedP0::gmsh_file` = ⟨*input file name* ⟩

Default: ⟨*obligatory* ⟩

Input file with ASCII GMSH file format.

`FieldInterpolatedP0::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: **IT::FieldElementwise** implements abstract type: IT::Field:R3 → Enum

---

R3 → Enum Field constant in space.

`FieldElementwise::TYPE` = ⟨*selection: Field:R3 → Enum_TYPE_selection* ⟩

Default: FieldElementwise

Sub-record selection.

`FieldElementwise::gmsh_file` = ⟨*input file name* ⟩

Default: ⟨*obligatory* ⟩

Input file with ASCII GMSH file format.

`FieldElementwise::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: **IT::Field:R3 → Real** default descendant: IT::FieldConstant

---

Descendants:

Abstract record for all time-space functions.

`IT::FieldConstant`

`IT::FieldFormula`

`IT::FieldPython`

`IT::FieldInterpolatedP0`

`IT::FieldElementwise`

---

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

---

R3 → Real Field constant in space.

`FieldConstant::TYPE` = ⟨*selection: Field:R3 → Real_TYPE_selection* ⟩
Default: FieldConstant
Sub-record selection.

`FieldConstant::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: **IT::FieldFormula** implements abstract type: IT::Field:R3 → Real

---

R3 → Real Field given by runtime interpreted formula.

`FieldFormula::TYPE` = ⟨*selection: Field:R3 → Real_TYPE_selection* ⟩
Default: FieldFormula
Sub-record selection.

`FieldFormula::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: **IT::FieldPython** implements abstract type: IT::Field:R3 → Real

---

R3 → Real Field given by a Python script.

`FieldPython::TYPE` = ⟨*selection: Field:R3 → Real_TYPE_selection* ⟩
Default: FieldPython
Sub-record selection.

`FieldPython::script_string` = ⟨*String (generic)* ⟩
Default: ⟨*value at read time* ⟩
Python script given as in place string

`FieldPython::script_file` = ⟨*input file name* ⟩
Default: Obligatory if 'script_striong' is not given.
Python script given as external file

`FieldPython::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: **IT::FieldInterpolatedP0** implements abstract type: IT::Field:R3 → Real

R3 → Real Field constant in space.

`FieldInterpolatedP0::TYPE` = ⟨*selection: Field:R3 → Real_TYPE_selection* ⟩
Default: FieldInterpolatedP0
Sub-record selection.

`FieldInterpolatedP0::gmsh_file` = ⟨*input file name* ⟩
Default: ⟨*obligatory* ⟩
Input file with ASCII GMSH file format.

`FieldInterpolatedP0::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: **IT::FieldElementwise** implements abstract type: IT::Field:R3 → Real

R3 → Real Field constant in space.

`FieldElementwise::TYPE` = ⟨*selection: Field:R3 → Real_TYPE_selection* ⟩
Default: FieldElementwise
Sub-record selection.

`FieldElementwise::gmsh_file` = ⟨*input file name* ⟩
Default: ⟨*obligatory* ⟩
Input file with ASCII GMSH file format.

`FieldElementwise::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: **IT::Field:R3 → Real** default descendant: IT::FieldConstant

Descendants:
Abstract record for all time-space functions.

`IT::FieldConstant`

`IT::FieldFormula`

`IT::FieldPython`

`IT::FieldInterpolatedP0`

`IT::FieldElementwise`

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

Mixed-Hybrid solver for unsteady saturated Darcy flow.

`Unsteady_MH::TYPE` = ⟨*selection: DarcyFlowMH_TYPE_selection* ⟩
Default: Unsteady_MH
Sub-record selection.

`Unsteady_MH::n_schurs` = ⟨*Integer[0, 2]* ⟩
Default: ⟨*2* ⟩
Number of Schur complements to perform when solving MH sytem.

`Unsteady_MH::solver` = ⟨*abstract type: IT::LinSys* ⟩
Default: ⟨*obligatory* ⟩
Linear solver for MH problem.

`Unsteady_MH::output` = ⟨*record: IT::DarcyMHOutput* ⟩
Default: ⟨*obligatory* ⟩
Parameters of output form MH module.

`Unsteady_MH::mortar_method` = ⟨*selection: IT::MH_MortarMethod* ⟩
Default: None
Method for coupling Darcy flow between dimensions.

`Unsteady_MH::balance` = ⟨*record: IT::Balance* ⟩
Default: ⟨*obligatory* ⟩
Settings for computing mass balance.

`Unsteady_MH::input_fields` = ⟨*Array of Record: IT::DarcyFlowMH_Data* ⟩
Default: ⟨*obligatory* ⟩

`Unsteady_MH::time` = ⟨*record: IT::TimeGovernor* ⟩
Default: ⟨*obligatory* ⟩
Time governor setting for the unsteady Darcy flow model.

---

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

Lumped Mixed-Hybrid solver for unsteady saturated Darcy flow.

`Unsteady_LMH::TYPE` = ⟨*selection: DarcyFlowMH_TYPE_selection* ⟩
Default: Unsteady_LMH
Sub-record selection.

`Unsteady_LMH::n_schurs` = ⟨*Integer[0, 2]* ⟩
Default: ⟨*2* ⟩
Number of Schur complements to perform when solving MH sytem.

`Unsteady_LMH::solver` = ⟨*abstract type: IT::LinSys* ⟩
Default: ⟨*obligatory* ⟩
Linear solver for MH problem.

`Unsteady_LMH::output` = ⟨*record: IT::DarcyMHOutput* ⟩
Default: ⟨*obligatory* ⟩
Parameters of output form MH module.

`Unsteady_LMH::mortar_method` = ⟨*selection: IT::MH_MortarMethod* ⟩
Default: None
Method for coupling Darcy flow between dimensions.

`Unsteady_LMH::balance` = ⟨*record: IT::Balance* ⟩
Default: ⟨*obligatory* ⟩
Settings for computing mass balance.

`Unsteady_LMH::input_fields` = ⟨*Array of Record: IT::DarcyFlowMH_Data* ⟩
Default: ⟨*obligatory* ⟩

`Unsteady_LMH::time` = ⟨*record: IT::TimeGovernor* ⟩
Default: ⟨*obligatory* ⟩
Time governor setting for the unsteady Darcy flow model.

---

abstract type: **IT::Transport**

Descendants:
Secondary equation for transport of substances.

`IT::TransportOperatorSplitting`

`IT::SoluteTransport_DG`

`IT::HeatTransfer_DG`

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

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

`TransportOperatorSplitting::TYPE` = ⟨*selection: Transport_TYPE_selection* ⟩
Default: TransportOperatorSplitting
Sub-record selection.

`TransportOperatorSplitting::time` = ⟨*record: IT::TimeGovernor* ⟩
Default: ⟨*obligatory* ⟩
Time governor setting for the secondary equation.

`TransportOperatorSplitting::balance` = ⟨*record: IT::Balance* ⟩
Default: ⟨*obligatory* ⟩
Settings for computing balance.

`TransportOperatorSplitting::output_stream` = ⟨*record: IT::OutputStream* ⟩
Default: ⟨*obligatory* ⟩
Parameters of output stream.

`TransportOperatorSplitting::substances` = ⟨*Array of Record: IT::Substance* ⟩
Default: ⟨*obligatory* ⟩
Specification of transported substances.

`TransportOperatorSplitting::reaction_term` = ⟨*abstract type: IT::ReactionTerm* ⟩
Default: ⟨*optional* ⟩
Reaction model involved in transport.

`TransportOperatorSplitting::input_fields` = ⟨*Array of Record: IT::TransportOperatorSplitting_Data*
⟩
Default: ⟨*obligatory* ⟩

`TransportOperatorSplitting::output_fields` = ⟨*Array of Selection: IT::ConvectionTransport_Output*
⟩
Default: ⟨*conc* ⟩
List of fields to write to output file.

record: **IT::Substance** constructible from key: Substance::name

Chemical substance.

`Substance::name` = ⟨*String (generic)* ⟩
Default: ⟨*obligatory* ⟩
Name of the substance.

`Substance::molar_mass` = ⟨*Double[0,]* ⟩
Default: ⟨*1* ⟩
Molar mass of the substance [kg/mol].

abstract type: **IT::ReactionTerm**

Descendants:
Equation for reading information about simple chemical reactions.

`IT::FirstOrderReaction`

`IT::RadioactiveDecay`

`IT::Sorption`
`IT::SorptionMobile`
`IT::SorptionImmobile`
`IT::DualPorosity`
`IT::Semchem`

---

record: **IT::FirstOrderReaction** implements abstract type: IT::ReactionTerm

---

A model of first order chemical reactions (decompositions of a reactant into products).

`FirstOrderReaction::TYPE` = ⟨*selection: ReactionTerm_TYPE_selection* ⟩
Default: FirstOrderReaction
Sub-record selection.

`FirstOrderReaction::reactions` = ⟨*Array of Record: IT::Reaction* ⟩
Default: ⟨*obligatory* ⟩
An array of first order chemical reactions.

`FirstOrderReaction::ode_solver` = ⟨*abstract type: IT::LinearODESolver* ⟩
Default: ⟨*optional* ⟩
Numerical solver for the system of first order ordinary differential equations coming from the model.

---

record: **IT::Reaction**

Describes a single first order chemical reaction.

`Reaction::reactants` = ⟨*Array [1,] of Record: IT::FirstOrderReactionReactant* ⟩
Default: ⟨*obligatory* ⟩
An array of reactants. Do not use array, reactions with only one reactant (decays) are implemented at the moment!

`Reaction::reaction_rate` = ⟨*Double[0,]* ⟩
Default: ⟨*obligatory* ⟩
The reaction rate coefficient of the first order reaction.

`Reaction::products` = ⟨*Array [1,] of Record: IT::FirstOrderReactionProduct* ⟩
Default: ⟨*obligatory* ⟩
An array of products.

---

record: **IT::FirstOrderReactionReactant** constructible from key: FirstOrderReactionReactant::name

---

A record describing a reactant of a reaction.

`FirstOrderReactionReactant::name` = ⟨*String (generic)* ⟩
Default: ⟨*obligatory* ⟩
The name of the reactant.

---

record: **IT::FirstOrderReactionProduct** constructible from key: FirstOrderReactionProduct::name

---

A record describing a product of a reaction.

`FirstOrderReactionProduct::name` = ⟨*String (generic)* ⟩
Default: ⟨*obligatory* ⟩
The name of the product.

`FirstOrderReactionProduct::branching_ratio` = ⟨*Double[0,]* ⟩
Default: ⟨*1.0* ⟩
The branching ratio of the product when there are more products.
The value must be positive. Further, the branching ratios of all products are normalized in order to sum to one.
The default value 1.0, should only be used in the case of single product.

abstract type: **IT::LinearODESolver**

Descendants:
Solver of a linear system of ODEs.

`IT::PadeApproximant`

`IT::LinearODEAnalytic`

record: **IT::PadeApproximant** implements abstract type: IT::LinearODESolver

Record with an information about pade approximant parameters.

`PadeApproximant::TYPE` = ⟨*selection: LinearODESolver_TYPE_selection* ⟩
Default: PadeApproximant
Sub-record selection.

`PadeApproximant::nominator_degree` = ⟨*Integer[1,]* ⟩
Default: ⟨*2* ⟩
Polynomial degree of the nominator of Pade approximant.

`PadeApproximant::denominator_degree` = ⟨*Integer[1,]* ⟩
Default: ⟨*2* ⟩
Polynomial degree of the nominator of Pade approximant

record: **IT::LinearODEAnalytic** implements abstract type: IT::LinearODESolver

Evaluate analytic solution of the system of ODEs.

`LinearODEAnalytic::TYPE` = ⟨*selection: LinearODESolver_TYPE_selection* ⟩
Default: LinearODEAnalytic
Sub-record selection.

record: **IT::RadioactiveDecay** implements abstract type: IT::ReactionTerm

A model of a radioactive decay and possibly of a decay chain.

`RadioactiveDecay::TYPE` = ⟨*selection: ReactionTerm_TYPE_selection* ⟩
Default: RadioactiveDecay
Sub-record selection.

`RadioactiveDecay::decays` = ⟨*Array [1,] of Record: IT::Decay* ⟩
Default: ⟨*obligatory* ⟩
An array of radioactive decays.

`RadioactiveDecay::ode_solver` = ⟨*abstract type: IT::LinearODESolver* ⟩
Default: ⟨*optional* ⟩
Numerical solver for the system of first order ordinary differential equations coming from the model.

record: **IT::Decay**

A model of a radioactive decay.

`Decay::radionuclide` = ⟨*String (generic)* ⟩
Default: ⟨*obligatory* ⟩
The name of the parent radionuclide.

`Decay::half_life` = ⟨*Double[0,]* ⟩
Default: ⟨*obligatory* ⟩
The half life of the parent radionuclide in seconds.

`Decay::products` = ⟨*Array [1,] of Record: IT::RadioactiveDecayProduct* ⟩
Default: ⟨*obligatory* ⟩
An array of the decay products (daughters).

---

record: **IT::RadioactiveDecayProduct** constructible from key: RadioactiveDecayProduct::name

---

A record describing a product of a radioactive decay.

`RadioactiveDecayProduct::name` = ⟨*String (generic)* ⟩
Default: ⟨*obligatory* ⟩
The name of the product.

`RadioactiveDecayProduct::energy` = ⟨*Double[0,]* ⟩
Default: ⟨*0.0* ⟩
Not used at the moment! The released energy in MeV from the decay of the radionuclide into the product.

`RadioactiveDecayProduct::branching_ratio` = ⟨*Double[0,]* ⟩
Default: ⟨*1.0* ⟩
The branching ratio of the product when there is more than one.Considering only one product, the default ratio 1.0 is used.Its value must be positive. Further, the branching ratios of all products are normalizedby their sum, so the sum then gives 1.0 (this also resolves possible rounding errors).

---

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

---

Sorption model in the reaction term of transport.

`Sorption::TYPE` = ⟨*selection: ReactionTerm_TYPE_selection* ⟩
Default: Sorption
Sub-record selection.

`Sorption::substances` = ⟨*Array [1,] of String (generic)* ⟩
Default: ⟨*obligatory* ⟩
Names of the substances that take part in the sorption model.

`Sorption::solvent_density` = ⟨*Double[0,]* ⟩
Default: ⟨*1.0* ⟩
Density of the solvent.

`Sorption::substeps` = ⟨*Integer[1,]* ⟩
Default: ⟨*1000* ⟩
Number of equidistant substeps, molar mass and isotherm intersections

`Sorption::solubility` = ⟨*Array of Double* ⟩
Default: ⟨*optional* ⟩
Specifies solubility limits of all the sorbing species.

`Sorption::table_limits` = ⟨*Array of Double* ⟩

Default: ⟨*optional* ⟩
Specifies highest aqueous concentration in interpolation table.

`Sorption::input_fields` = ⟨*Array of Record: IT::Sorption_Data* ⟩
Default: ⟨*obligatory* ⟩
Containes region specific data necessary to construct isotherms.

`Sorption::reaction_liquid` = ⟨*abstract type: IT::ReactionTerm* ⟩
Default: ⟨*optional* ⟩
Reaction model following the sorption in the liquid.

`Sorption::reaction_solid` = ⟨*abstract type: IT::ReactionTerm* ⟩
Default: ⟨*optional* ⟩
Reaction model following the sorption in the solid.

`Sorption::output_fields` = ⟨*Array of Selection: IT::Sorption_Output* ⟩
Default: ⟨*conc_solid* ⟩
List of fields to write to output stream.

---

record: **IT::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.

`Sorption_Data::r_set` = ⟨*String (generic)* ⟩
Default: ⟨*optional* ⟩
Name of region set where to set fields.

`Sorption_Data::region` = ⟨*String (generic)* ⟩
Default: ⟨*optional* ⟩
Label of the region where to set fields.

`Sorption_Data::rid` = ⟨*Integer[0,]* ⟩
Default: ⟨*optional* ⟩
ID of the region where to set fields.

`Sorption_Data::time` = ⟨*Double[0,]* ⟩
Default: ⟨*0.0* ⟩
Apply field setting in this record after this time.
These times have to form an increasing sequence.

`Sorption_Data::rock_density` = ⟨*abstract type: IT::Field:R3 → Real* ⟩
Default: ⟨*optional* ⟩
Rock matrix density. $[m^{-3}kg]$

`Sorption_Data::sorption_type` = ⟨*abstract type: IT::Field:R3 → Enum[n]* ⟩
Default: ⟨*optional* ⟩
Considered sorption is described by selected isotherm. If porosity on an element is equal or even higher than 1.0 (meaning no sorbing surface), then type 'none' will be selected automatically. $[-]$

`Sorption_Data::isotherm_mult` = ⟨*abstract type: IT::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. [$kg^{-1}mol$]

`Sorption_Data::isotherm_other` = ⟨*abstract type: IT::Field:R3 → Real[n]* ⟩
Default: ⟨*optional* ⟩
Second parameters (alpha, ...) defining isotherm c_s = omega * (alpha*c_a)/(1-alpha*c_a). [−]

`Sorption_Data::init_conc_solid` = ⟨*abstract type: IT::Field:R3 → Real[n]* ⟩
Default: ⟨*optional* ⟩
Initial solid concentration of substances. Vector, one value for every substance. [$kg^{-1}mol$]

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

Descendants:
Abstract record for all time-space functions.

`IT::FieldConstant`

`IT::FieldFormula`

`IT::FieldPython`

`IT::FieldInterpolatedP0`

`IT::FieldElementwise`

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

R3 → Enum[n] Field constant in space.

`FieldConstant::TYPE` = ⟨*selection: Field:R3 → Enum[n]_TYPE_selection* ⟩
Default: FieldConstant
Sub-record selection.

`FieldConstant::value` = ⟨*Array [1,] of Selection: IT::SorptionType* ⟩
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: **IT::SorptionType**

Possible values:

`none` : No sorption 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: **IT::FieldFormula** implements abstract type: IT::Field:R3 → Enum[n]

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

`FieldFormula::TYPE` = ⟨*selection: Field:R3 → Enum[n]_TYPE_selection* ⟩

Default: FieldFormula

Sub-record selection.

`FieldFormula::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: **IT::FieldPython** implements abstract type: IT::Field:R3 → Enum[n]

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

`FieldPython::TYPE` = ⟨*selection: Field:R3 → Enum[n]_TYPE_selection* ⟩

Default: FieldPython

Sub-record selection.

`FieldPython::script_string` = ⟨*String (generic)* ⟩

Default: ⟨*value at read time* ⟩

Python script given as in place string

`FieldPython::script_file` = ⟨*input file name* ⟩

Default: Obligatory if 'script_striong' is not given.

Python script given as external file

`FieldPython::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: **IT::FieldInterpolatedP0** implements abstract type: IT::Field:R3 → Enum[n]

R3 → Enum[n] Field constant in space.

`FieldInterpolatedP0::TYPE` = ⟨*selection: Field:R3 → Enum[n]_TYPE_selection* ⟩

Default: FieldInterpolatedP0

Sub-record selection.

`FieldInterpolatedP0::gmsh_file` = ⟨*input file name* ⟩

Default: ⟨*obligatory* ⟩

Input file with ASCII GMSH file format.

`FieldInterpolatedP0::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: **IT::FieldElementwise** implements abstract type: IT::Field:R3 → Enum[n]

R3 → Enum[n] Field constant in space.

`FieldElementwise::TYPE` = ⟨*selection: Field:R3 → Enum[n]_TYPE_selection* ⟩
Default: FieldElementwise
Sub-record selection.

`FieldElementwise::gmsh_file` = ⟨*input file name* ⟩
Default: ⟨*obligatory* ⟩
Input file with ASCII GMSH file format.

`FieldElementwise::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: **IT::Field:R3 → Real[n]** default descendant: IT::FieldConstant

Descendants:
Abstract record for all time-space functions.

`IT::FieldConstant`

`IT::FieldPython`

`IT::FieldFormula`

`IT::FieldElementwise`

`IT::FieldInterpolatedP0`

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

R3 → Real[n] Field constant in space.

`FieldConstant::TYPE` = ⟨*selection: Field:R3 → Real[n]_TYPE_selection* ⟩
Default: FieldConstant
Sub-record selection.

`FieldConstant::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: **IT::FieldPython** implements abstract type: IT::Field:R3 → Real[n]

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

`FieldPython::TYPE` = ⟨*selection: Field:R3 → Real[n]_TYPE_selection* ⟩
Default: FieldPython
Sub-record selection.

`FieldPython::script_string` = ⟨*String (generic)* ⟩
Default: ⟨*value at read time* ⟩
Python script given as in place string

`FieldPython::script_file` = ⟨*input file name* ⟩

Default: Obligatory if 'script_striong' is not given.

Python script given as external file

`FieldPython::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: **IT::FieldFormula** implements abstract type: IT::Field:R3 → Real[n]

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

`FieldFormula::TYPE` = ⟨*selection: Field:R3 → Real[n]_TYPE_selection* ⟩

Default: FieldFormula

Sub-record selection.

`FieldFormula::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: **IT::FieldElementwise** implements abstract type: IT::Field:R3 → Real[n]

R3 → Real[n] Field constant in space.

`FieldElementwise::TYPE` = ⟨*selection: Field:R3 → Real[n]_TYPE_selection* ⟩

Default: FieldElementwise

Sub-record selection.

`FieldElementwise::gmsh_file` = ⟨*input file name* ⟩

Default: ⟨*obligatory* ⟩

Input file with ASCII GMSH file format.

`FieldElementwise::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: **IT::FieldInterpolatedP0** implements abstract type: IT::Field:R3 → Real[n]

R3 → Real[n] Field constant in space.

`FieldInterpolatedP0::TYPE` = ⟨*selection: Field:R3 → Real[n]_TYPE_selection* ⟩

Default: FieldInterpolatedP0

Sub-record selection.

`FieldInterpolatedP0::gmsh_file` = ⟨*input file name* ⟩

Default: ⟨*obligatory* ⟩

Input file with ASCII GMSH file format.

`FieldInterpolatedP0::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: **IT::Sorption_Output**

Possible values:

`rock_density` : Output of the field rock_density [$m^{-3}kg$] (Rock matrix density.).

`sorption_type` : Output of the field sorption_type [$-$] (Considered sorption is described by selected isotherm. If porosity on an element is equal or even higher than 1.0 (meaning no sorbing surface), then type 'none' will be selected automatically.).

`isotherm_mult` : Output of the field isotherm_mult [$kg^{-1}mol$] (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 [$kg^{-1}mol$] (Initial solid concentration of substances. Vector, one value for every substance.).

`conc_solid` : Output of the field conc_solid [$m^{-3}kg$].

---

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

---

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

`SorptionMobile::TYPE` = ⟨*selection: ReactionTerm_TYPE_selection* ⟩
Default: SorptionMobile
Sub-record selection.

`SorptionMobile::substances` = ⟨*Array [1,] of String (generic)* ⟩
Default: ⟨*obligatory* ⟩
Names of the substances that take part in the sorption model.

`SorptionMobile::solvent_density` = ⟨*Double[0,]* ⟩
Default: ⟨*1.0* ⟩
Density of the solvent.

`SorptionMobile::substeps` = ⟨*Integer[1,]* ⟩
Default: ⟨*1000* ⟩
Number of equidistant substeps, molar mass and isotherm intersections

`SorptionMobile::solubility` = ⟨*Array of Double* ⟩
Default: ⟨*optional* ⟩
Specifies solubility limits of all the sorbing species.

`SorptionMobile::table_limits` = ⟨*Array of Double* ⟩
Default: ⟨*optional* ⟩
Specifies highest aqueous concentration in interpolation table.

`SorptionMobile::input_fields` = ⟨*Array of Record: IT::Sorption_Data* ⟩
Default: ⟨*obligatory* ⟩
Containes region specific data necessary to construct isotherms.

`SorptionMobile::reaction_liquid` = ⟨*abstract type: IT::ReactionTerm* ⟩
Default: ⟨*optional* ⟩

Reaction model following the sorption in the liquid.

`SorptionMobile::reaction_solid` = ⟨*abstract type: IT::ReactionTerm* ⟩
Default: ⟨*optional* ⟩
Reaction model following the sorption in the solid.

`SorptionMobile::output_fields` = ⟨*Array of Selection: IT::SorptionMobile_Output* ⟩
Default: ⟨*conc_solid* ⟩
List of fields to write to output stream.

---

selection type: **IT::SorptionMobile_Output**

Possible values:

`rock_density` : Output of the field rock_density [$m^{-3}kg$] (Rock matrix density.).

`sorption_type` : Output of the field sorption_type [$-$] (Considered sorption is described by selected isotherm. If porosity on an element is equal or even higher than 1.0 (meaning no sorbing surface), then type 'none' will be selected automatically.).

`isotherm_mult` : Output of the field isotherm_mult [$kg^{-1}mol$] (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 [$kg^{-1}mol$] (Initial solid concentration of substances. Vector, one value for every substance.).

`conc_solid` : Output of the field conc_solid [$m^{-3}kg$].

---

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

---

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

`SorptionImmobile::TYPE` = ⟨*selection: ReactionTerm_TYPE_selection* ⟩
Default: SorptionImmobile
Sub-record selection.

`SorptionImmobile::substances` = ⟨*Array [1,] of String (generic)* ⟩
Default: ⟨*obligatory* ⟩
Names of the substances that take part in the sorption model.

`SorptionImmobile::solvent_density` = ⟨*Double[0,]* ⟩
Default: ⟨*1.0* ⟩
Density of the solvent.

`SorptionImmobile::substeps` = ⟨*Integer[1,]* ⟩
Default: ⟨*1000* ⟩
Number of equidistant substeps, molar mass and isotherm intersections

`SorptionImmobile::solubility` = ⟨*Array of Double* ⟩
Default: ⟨*optional* ⟩
Specifies solubility limits of all the sorbing species.

`SorptionImmobile::table_limits` = ⟨*Array of Double* ⟩
Default: ⟨*optional* ⟩
Specifies highest aqueous concentration in interpolation table.

`SorptionImmobile::input_fields` = ⟨*Array of Record: IT::Sorption_Data* ⟩
Default: ⟨*obligatory* ⟩
Containes region specific data necessary to construct isotherms.

`SorptionImmobile::reaction_liquid` = ⟨*abstract type: IT::ReactionTerm* ⟩
Default: ⟨*optional* ⟩
Reaction model following the sorption in the liquid.

`SorptionImmobile::reaction_solid` = ⟨*abstract type: IT::ReactionTerm* ⟩
Default: ⟨*optional* ⟩
Reaction model following the sorption in the solid.

`SorptionImmobile::output_fields` = ⟨*Array of Selection: IT::SorptionImmobile_Output* ⟩
Default: ⟨*conc_immobile_solid* ⟩
List of fields to write to output stream.

---

selection type: **IT::SorptionImmobile_Output**

Possible values:

`rock_density` : Output of the field rock_density $[m^{-3}kg]$ (Rock matrix density.).

`sorption_type` : Output of the field sorption_type $[-]$ (Considered sorption is described by selected isotherm. If porosity on an element is equal or even higher than 1.0 (meaning no sorbing surface), then type 'none' will be selected automatically.).

`isotherm_mult` : Output of the field isotherm_mult $[kg^{-1}mol]$ (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 $[kg^{-1}mol]$ (Initial solid concentration of substances. Vector, one value for every substance.).

`conc_immobile_solid` : Output of the field conc_immobile_solid $[m^{-3}kg]$.

---

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

---

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

`DualPorosity::TYPE` = ⟨*selection: ReactionTerm_TYPE_selection* ⟩
Default: DualPorosity
Sub-record selection.

`DualPorosity::input_fields` = ⟨*Array of Record: IT::DualPorosity_Data* ⟩
Default: ⟨*obligatory* ⟩
Containes region specific data necessary to construct dual porosity model.

`DualPorosity::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).

`DualPorosity::reaction_mobile` = ⟨*abstract type: IT::ReactionTerm* ⟩
Default: ⟨*optional* ⟩
Reaction model in mobile zone.

`DualPorosity::reaction_immobile` = ⟨*abstract type: IT::ReactionTerm* ⟩
Default: ⟨*optional* ⟩
Reaction model in immobile zone.

`DualPorosity::output_fields` = ⟨*Array of Selection: IT::DualPorosity_Output* ⟩
Default: ⟨*conc_immobile* ⟩
List of fields to write to output stream.

---

record: **IT::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.

`DualPorosity_Data::r_set` = ⟨*String (generic)* ⟩
Default: ⟨*optional* ⟩
Name of region set where to set fields.

`DualPorosity_Data::region` = ⟨*String (generic)* ⟩
Default: ⟨*optional* ⟩
Label of the region where to set fields.

`DualPorosity_Data::rid` = ⟨*Integer[0,]* ⟩
Default: ⟨*optional* ⟩
ID of the region where to set fields.

`DualPorosity_Data::time` = ⟨*Double[0,]* ⟩
Default: ⟨*0.0* ⟩
Apply field setting in this record after this time.
These times have to form an increasing sequence.

`DualPorosity_Data::diffusion_rate_immobile` = ⟨*abstract type: IT::Field:R3 → Real[n]* ⟩
Default: ⟨*optional* ⟩
Diffusion coefficient of non-equilibrium linear exchange between mobile and immobile zone. $[s^{-1}]$

`DualPorosity_Data::porosity_immobile` = ⟨*abstract type: IT::Field:R3 → Real* ⟩
Default: ⟨*optional* ⟩
Porosity of the immobile zone. $[-]$

`DualPorosity_Data::init_conc_immobile` = ⟨*abstract type: IT::Field:R3 → Real[n]* ⟩
Default: ⟨*optional* ⟩
Initial concentration of substances in the immobile zone. $[m^{-3}kg]$

---

selection type: **IT::DualPorosity_Output**

Possible values:

`diffusion_rate_immobile` : Output of the field diffusion_rate_immobile $[s^{-1}]$ (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 $[m^{-3}kg]$ (Initial concentration of substances in the immobile zone.).

`conc_immobile` : Output of the field conc_immobile $[m^{-3}kg]$.

record: **IT::Semchem** implements abstract type: IT::ReactionTerm

Declares infos valid for all reactions. NOT SUPPORTED!!!.

`Semchem::TYPE` = ⟨*selection: ReactionTerm_TYPE_selection* ⟩
Default: Semchem
Sub-record selection.

`Semchem::precision` = ⟨*Integer[-2147483648,]* ⟩
Default: ⟨*obligatory* ⟩
How accurate should the simulation be, decimal places(?).

`Semchem::temperature` = ⟨*Double* ⟩
Default: ⟨*obligatory* ⟩
Isothermal reaction, thermodynamic temperature.

`Semchem::temp_gf` = ⟨*Double* ⟩
Default: ⟨*obligatory* ⟩
Thermodynamic parameter.

`Semchem::param_afi` = ⟨*Double* ⟩
Default: ⟨*obligatory* ⟩
Thermodynamic parameter.

`Semchem::param_b` = ⟨*Double* ⟩
Default: ⟨*obligatory* ⟩
Thermodynamic parameter.

`Semchem::epsilon` = ⟨*Double* ⟩
Default: ⟨*obligatory* ⟩
Thermodynamic parameter.

`Semchem::time_steps` = ⟨*Integer[-2147483648,]* ⟩
Default: ⟨*obligatory* ⟩
Simulation parameter.

`Semchem::slow_kinetic_steps` = ⟨*Integer[-2147483648,]* ⟩
Default: ⟨*obligatory* ⟩
Simulation parameter.

record: **IT::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.

`TransportOperatorSplitting_Data::r_set` = ⟨*String (generic)* ⟩
Default: ⟨*optional* ⟩
Name of region set where to set fields.

`TransportOperatorSplitting_Data::region` = ⟨*String (generic)* ⟩
Default: ⟨*optional* ⟩
Label of the region where to set fields.

`TransportOperatorSplitting_Data::rid` = ⟨*Integer[0,]* ⟩
Default: ⟨*optional* ⟩

ID of the region where to set fields.

`TransportOperatorSplitting_Data::time` = ⟨*Double[0,]* ⟩
Default: ⟨*0.0* ⟩
Apply field setting in this record after this time.
These times have to form an increasing sequence.

`TransportOperatorSplitting_Data::porosity` = ⟨*abstract type: IT::Field:R3 → Real* ⟩
Default: ⟨*optional* ⟩
Mobile porosity [−]

`TransportOperatorSplitting_Data::sources_density` = ⟨*abstract type: IT::Field:R3 → Real[n]* ⟩
Default: ⟨*optional* ⟩
Density of concentration sources. $[m^{-3}kgs^{-1}]$

`TransportOperatorSplitting_Data::sources_sigma` = ⟨*abstract type: IT::Field:R3 → Real[n]* ⟩
Default: ⟨*optional* ⟩
Concentration flux. $[s^{-1}]$

`TransportOperatorSplitting_Data::sources_conc` = ⟨*abstract type: IT::Field:R3 → Real[n]* ⟩
Default: ⟨*optional* ⟩
Concentration sources threshold. $[m^{-3}kg]$

`TransportOperatorSplitting_Data::bc_conc` = ⟨*abstract type: IT::Field:R3 → Real[n]* ⟩
Default: ⟨*optional* ⟩
Boundary conditions for concentrations. $[m^{-3}kg]$

`TransportOperatorSplitting_Data::init_conc` = ⟨*abstract type: IT::Field:R3 → Real[n]* ⟩
Default: ⟨*optional* ⟩
Initial concentrations. $[m^{-3}kg]$

`TransportOperatorSplitting_Data::transport_old_bcd_file` = ⟨*input file name* ⟩
Default: ⟨*optional* ⟩
File with mesh dependent boundary conditions (obsolete).

selection type: **IT::ConvectionTransport_Output**

Possible values:

`porosity` : Output of the field porosity [−] (Mobile porosity).

`sources_density` : Output of the field sources_density $[m^{-3}kgs^{-1}]$ (Density of concentration sources.).

`sources_sigma` : Output of the field sources_sigma $[s^{-1}]$ (Concentration flux.).

`sources_conc` : Output of the field sources_conc $[m^{-3}kg]$ (Concentration sources threshold.).

`init_conc` : Output of the field init_conc $[m^{-3}kg]$ (Initial concentrations.).

`conc` : Output of the field conc $[m^{-3}kg]$.

`region_id` : Output of the field region_id [−].

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

DG solver for solute transport.

`SoluteTransport_DG::TYPE` = ⟨*selection: Transport_TYPE_selection* ⟩
Default: SoluteTransport_DG
Sub-record selection.

`SoluteTransport_DG::time` = ⟨*record: IT::TimeGovernor* ⟩
Default: ⟨*obligatory* ⟩
Time governor setting for the secondary equation.

`SoluteTransport_DG::balance` = ⟨*record: IT::Balance* ⟩
Default: ⟨*obligatory* ⟩
Settings for computing balance.

`SoluteTransport_DG::output_stream` = ⟨*record: IT::OutputStream* ⟩
Default: ⟨*obligatory* ⟩
Parameters of output stream.

`SoluteTransport_DG::substances` = ⟨*Array of Record: IT::Substance* ⟩
Default: ⟨*obligatory* ⟩
Names of transported substances.

`SoluteTransport_DG::solver` = ⟨*record: IT::Petsc* ⟩
Default: ⟨*obligatory* ⟩
Linear solver for MH problem.

`SoluteTransport_DG::input_fields` = ⟨*Array of Record: IT::SoluteTransport_DG_Data*
⟩
Default: ⟨*obligatory* ⟩

`SoluteTransport_DG::dg_variant` = ⟨*selection: IT::DG_variant* ⟩
Default: non-symmetric
Variant of interior penalty discontinuous Galerkin method.

`SoluteTransport_DG::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).

`SoluteTransport_DG::output_fields` = ⟨*Array of Selection: IT::SoluteTransport_DG_Output*
⟩
Default: ⟨*conc* ⟩
List of fields to write to output file.

---

record: **IT::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.

`SoluteTransport_DG_Data::r_set` = ⟨*String (generic)* ⟩
Default: ⟨*optional* ⟩
Name of region set where to set fields.

`SoluteTransport_DG_Data::region` = ⟨*String (generic)* ⟩
Default: ⟨*optional* ⟩

Label of the region where to set fields.

`SoluteTransport_DG_Data::rid` = ⟨*Integer[0,]*⟩
Default: ⟨*optional*⟩
ID of the region where to set fields.

`SoluteTransport_DG_Data::time` = ⟨*Double[0,]*⟩
Default: ⟨*0.0*⟩
Apply field setting in this record after this time.
These times have to form an increasing sequence.

`SoluteTransport_DG_Data::porosity` = ⟨*abstract type: IT::Field:R3 → Real*⟩
Default: ⟨*optional*⟩
Mobile porosity $[-]$

`SoluteTransport_DG_Data::sources_density` = ⟨*abstract type: IT::Field:R3 → Real[n]*
⟩
Default: ⟨*optional*⟩
Density of concentration sources. $[m^{-3}kgs^{-1}]$

`SoluteTransport_DG_Data::sources_sigma` = ⟨*abstract type: IT::Field:R3 → Real[n]*
⟩
Default: ⟨*optional*⟩
Concentration flux. $[s^{-1}]$

`SoluteTransport_DG_Data::sources_conc` = ⟨*abstract type: IT::Field:R3 → Real[n]*⟩
Default: ⟨*optional*⟩
Concentration sources threshold. $[m^{-3}kg]$

`SoluteTransport_DG_Data::bc_conc` = ⟨*abstract type: IT::Field:R3 → Real[n]*⟩
Default: ⟨*optional*⟩
Dirichlet boundary condition (for each substance). $[m^{-3}kg]$

`SoluteTransport_DG_Data::init_conc` = ⟨*abstract type: IT::Field:R3 → Real[n]*⟩
Default: ⟨*optional*⟩
Initial concentrations. $[m^{-3}kg]$

`SoluteTransport_DG_Data::disp_l` = ⟨*abstract type: IT::Field:R3 → Real[n]*⟩
Default: ⟨*optional*⟩
Longitudal dispersivity (for each substance). $[m]$

`SoluteTransport_DG_Data::disp_t` = ⟨*abstract type: IT::Field:R3 → Real[n]*⟩
Default: ⟨*optional*⟩
Transversal dispersivity (for each substance). $[m]$

`SoluteTransport_DG_Data::diff_m` = ⟨*abstract type: IT::Field:R3 → Real[n]*⟩
Default: ⟨*optional*⟩
Molecular diffusivity (for each substance). $[m^2s^{-1}]$

`SoluteTransport_DG_Data::fracture_sigma` = ⟨*abstract type: IT::Field:R3 → Real[n]*
⟩
Default: ⟨*optional*⟩
Coefficient of diffusive transfer through fractures (for each substance). $[-]$

`SoluteTransport_DG_Data::dg_penalty` = ⟨*abstract type: IT::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. $[-]$

`SoluteTransport_DG_Data::bc_type` = ⟨*abstract type: IT::Field:R3 → Enum[n]*⟩

Default: ⟨*optional* ⟩
Boundary condition type, possible values: inflow, dirichlet, neumann, robin. [–]

`SoluteTransport_DG_Data::bc_flux` = ⟨*abstract type: IT::Field:R3 → Real[n]* ⟩
Default: ⟨*optional* ⟩
Flux in Neumann boundary condition. $[m^{1-d}kgs^{-1}]$

`SoluteTransport_DG_Data::bc_robin_sigma` = ⟨*abstract type: IT::Field:R3 → Real[n]* ⟩
Default: ⟨*optional* ⟩
Conductivity coefficient in Robin boundary condition. $[m^{4-d}s^{-1}]$

---

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

---

Descendants:
Abstract record for all time-space functions.

`IT::FieldConstant`

`IT::FieldFormula`

`IT::FieldPython`

`IT::FieldInterpolatedP0`

`IT::FieldElementwise`

---

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

---

R3 → Enum[n] Field constant in space.

`FieldConstant::TYPE` = ⟨*selection: Field:R3 → Enum[n]_TYPE_selection* ⟩
Default: FieldConstant
Sub-record selection.

`FieldConstant::value` = ⟨*Array [1,] of Selection: IT::TransportDG_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: **IT::TransportDG_BC_Type**

Types of boundary condition supported by the transport DG model (solute transport or heat transfer).
Possible values:

`none` : Homogeneous Neumann boundary condition. Zero flux

`dirichlet` : Dirichlet boundary condition.

`neumann` : Neumann boundary condition. Prescribe water outflow by the 'bc_flux' field.

`robin` : Robin boundary condition. Water outflow equal to $sigma(h - h^R)$.

`inflow` : Prescribes the concentration in the inflow water on the inflow part of the boundary.

---

selection type: **IT::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: **IT::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 $[m^{-3}kgs^{-1}]$ (Density of concentration sources.).

`sources_sigma` : Output of the field sources_sigma $[s^{-1}]$ (Concentration flux.).

`sources_conc` : Output of the field sources_conc $[m^{-3}kg]$ (Concentration sources threshold.).

`init_conc` : Output of the field init_conc $[m^{-3}kg]$ (Initial concentrations.).

`disp_l` : Output of the field disp_l $[m]$ (Longitudal dispersivity (for each substance).).

`disp_t` : Output of the field disp_t $[m]$ (Transversal dispersivity (for each substance).).

`diff_m` : Output of the field diff_m $[m^2s^{-1}]$ (Molecular diffusivity (for each substance).).

`conc` : Output of the field conc $[m^{-3}kg]$.

`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.).

`region_id` : Output of the field region_id $[-]$.

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

DG solver for heat transfer.

`HeatTransfer_DG::TYPE` = ⟨*selection: Transport_TYPE_selection* ⟩
Default: HeatTransfer_DG
Sub-record selection.

`HeatTransfer_DG::time` = ⟨*record: IT::TimeGovernor* ⟩
Default: ⟨*obligatory* ⟩
Time governor setting for the secondary equation.

`HeatTransfer_DG::balance` = ⟨*record: IT::Balance* ⟩
Default: ⟨*obligatory* ⟩
Settings for computing balance.

`HeatTransfer_DG::output_stream` = ⟨*record: IT::OutputStream* ⟩
Default: ⟨*obligatory* ⟩
Parameters of output stream.

`HeatTransfer_DG::solver` = ⟨*record: IT::Petsc* ⟩
Default: ⟨*obligatory* ⟩
Linear solver for MH problem.

`HeatTransfer_DG::input_fields` = ⟨*Array of Record: IT::HeatTransfer_DG_Data* ⟩
Default: ⟨*obligatory* ⟩

`HeatTransfer_DG::dg_variant` = ⟨*selection: IT::DG_variant* ⟩
Default: non-symmetric
Variant of interior penalty discontinuous Galerkin method.

`HeatTransfer_DG::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).

`HeatTransfer_DG::output_fields` = ⟨*Array of Selection: IT::HeatTransfer_DG_Output* ⟩
Default: ⟨*temperature* ⟩
List of fields to write to output file.

---

record: **IT::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.

`HeatTransfer_DG_Data::r_set` = ⟨*String (generic)* ⟩
Default: ⟨*optional* ⟩
Name of region set where to set fields.

`HeatTransfer_DG_Data::region` = ⟨*String (generic)* ⟩
Default: ⟨*optional* ⟩
Label of the region where to set fields.

`HeatTransfer_DG_Data::rid` = ⟨*Integer[0,]* ⟩
Default: ⟨*optional* ⟩
ID of the region where to set fields.

`HeatTransfer_DG_Data::time` = ⟨*Double[0,]* ⟩
Default: ⟨*0.0* ⟩
Apply field setting in this record after this time.
These times have to form an increasing sequence.

`HeatTransfer_DG_Data::bc_temperature` = ⟨*abstract type: IT::Field:R3 → Real* ⟩
Default: ⟨*optional* ⟩
Boundary value of temperature. $[K]$

`HeatTransfer_DG_Data::init_temperature` = ⟨*abstract type: IT::Field:R3 → Real* ⟩
Default: ⟨*optional* ⟩
Initial temperature. $[K]$

`HeatTransfer_DG_Data::porosity` = ⟨*abstract type: IT::Field:R3 → Real* ⟩
Default: ⟨*optional* ⟩
Porosity. $[-]$

`HeatTransfer_DG_Data::fluid_density` = ⟨*abstract type: IT::Field:R3 → Real* ⟩
Default: ⟨*optional* ⟩
Density of fluid. $[m^{-3}kg]$

`HeatTransfer_DG_Data::fluid_heat_capacity` = ⟨*abstract type: IT::Field:R3 → Real* ⟩

Default: ⟨*optional* ⟩

Heat capacity of fluid. $[m^2 s^{-2} K^{-1}]$

`HeatTransfer_DG_Data::fluid_heat_conductivity` = ⟨*abstract type: IT::Field:R3 → Real* ⟩

Default: ⟨*optional* ⟩

Heat conductivity of fluid. $[mkgs^{-3} K^{-1}]$

`HeatTransfer_DG_Data::solid_density` = ⟨*abstract type: IT::Field:R3 → Real* ⟩

Default: ⟨*optional* ⟩

Density of solid (rock). $[m^{-3} kg]$

`HeatTransfer_DG_Data::solid_heat_capacity` = ⟨*abstract type: IT::Field:R3 → Real* ⟩

Default: ⟨*optional* ⟩

Heat capacity of solid (rock). $[m^2 s^{-2} K^{-1}]$

`HeatTransfer_DG_Data::solid_heat_conductivity` = ⟨*abstract type: IT::Field:R3 → Real* ⟩

Default: ⟨*optional* ⟩

Heat conductivity of solid (rock). $[mkgs^{-3} K^{-1}]$

`HeatTransfer_DG_Data::disp_l` = ⟨*abstract type: IT::Field:R3 → Real* ⟩

Default: ⟨*optional* ⟩

Longitudal heat dispersivity in fluid. $[m]$

`HeatTransfer_DG_Data::disp_t` = ⟨*abstract type: IT::Field:R3 → Real* ⟩

Default: ⟨*optional* ⟩

Transversal heat dispersivity in fluid. $[m]$

`HeatTransfer_DG_Data::fluid_thermal_source` = ⟨*abstract type: IT::Field:R3 → Real* ⟩

Default: ⟨*optional* ⟩

Thermal source density in fluid. $[m^{-1} kgs^{-3}]$

`HeatTransfer_DG_Data::solid_thermal_source` = ⟨*abstract type: IT::Field:R3 → Real* ⟩

Default: ⟨*optional* ⟩

Thermal source density in solid. $[m^{-1} kgs^{-3}]$

`HeatTransfer_DG_Data::fluid_heat_exchange_rate` = ⟨*abstract type: IT::Field:R3 → Real* ⟩

Default: ⟨*optional* ⟩

Heat exchange rate in fluid. $[s^{-1}]$

`HeatTransfer_DG_Data::solid_heat_exchange_rate` = ⟨*abstract type: IT::Field:R3 → Real* ⟩

Default: ⟨*optional* ⟩

Heat exchange rate of source in solid. $[s^{-1}]$

`HeatTransfer_DG_Data::fluid_ref_temperature` = ⟨*abstract type: IT::Field:R3 → Real* ⟩

Default: ⟨*optional* ⟩

Reference temperature of source in fluid. $[K]$

`HeatTransfer_DG_Data::solid_ref_temperature` = ⟨*abstract type: IT::Field:R3 → Real* ⟩

Default: ⟨*optional* ⟩
Reference temperature in solid. $[K]$

`HeatTransfer_DG_Data::fracture_sigma` = ⟨*abstract type: IT::Field:R3 → Real[n]* ⟩
Default: ⟨*optional* ⟩
Coefficient of diffusive transfer through fractures (for each substance). $[-]$

`HeatTransfer_DG_Data::dg_penalty` = ⟨*abstract type: IT::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. $[-]$

`HeatTransfer_DG_Data::bc_type` = ⟨*abstract type: IT::Field:R3 → Enum[n]* ⟩
Default: ⟨*optional* ⟩
Boundary condition type, possible values: inflow, dirichlet, neumann, robin. $[-]$

`HeatTransfer_DG_Data::bc_flux` = ⟨*abstract type: IT::Field:R3 → Real[n]* ⟩
Default: ⟨*optional* ⟩
Flux in Neumann boundary condition. $[m^{1-d}kgs^{-1}]$

`HeatTransfer_DG_Data::bc_robin_sigma` = ⟨*abstract type: IT::Field:R3 → Real[n]* ⟩
Default: ⟨*optional* ⟩
Conductivity coefficient in Robin boundary condition. $[m^{4-d}s^{-1}]$

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

Descendants:
Abstract record for all time-space functions.

`IT::FieldConstant`
`IT::FieldFormula`
`IT::FieldPython`
`IT::FieldInterpolatedP0`
`IT::FieldElementwise`

selection type: **IT::HeatTransfer_DG_Output**

Selection for output fields of DG solver for heat transfer.
Possible values:

`init_temperature` : Output of the field init_temperature $[K]$ (Initial temperature.).
`porosity` : Output of the field porosity $[-]$ (Porosity.).
`fluid_density` : Output of the field fluid_density $[m^{-3}kg]$ (Density of fluid.).
`fluid_heat_capacity` : Output of the field fluid_heat_capacity $[m^2s^{-2}K^{-1}]$ (Heat capacity of fluid.).
`fluid_heat_conductivity` : Output of the field fluid_heat_conductivity $[mkgs^{-3}K^{-1}]$ (Heat conductivity of fluid.).
`solid_density` : Output of the field solid_density $[m^{-3}kg]$ (Density of solid (rock).).
`solid_heat_capacity` : Output of the field solid_heat_capacity $[m^2s^{-2}K^{-1}]$ (Heat capacity of solid (rock).).
`solid_heat_conductivity` : Output of the field solid_heat_conductivity $[mkgs^{-3}K^{-1}]$ (Heat conductivity of solid (rock).).
`disp_l` : Output of the field disp_l $[m]$ (Longitudal heat dispersivity in fluid.).

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

`fluid_thermal_source` : Output of the field fluid_thermal_source [$m^{-1}kgs^{-3}$] (Thermal source density in fluid.).

`solid_thermal_source` : Output of the field solid_thermal_source [$m^{-1}kgs^{-3}$] (Thermal source density in solid.).

`fluid_heat_exchange_rate` : Output of the field fluid_heat_exchange_rate [s^{-1}] (Heat exchange rate in fluid.).

`solid_heat_exchange_rate` : Output of the field solid_heat_exchange_rate [s^{-1}] (Heat exchange rate of source in solid.).

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

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

`temperature` : Output of the field temperature [K].

`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.).

`region_id` : Output of the field region_id [$-$].

0.1 Simple md support

showcase of **strong** and *em* fonts (or ***both***), links to Google

Simple *lists*:

- hi
- hello
- greetings

Also numbered:

1. foo
2. bar
3. stuff

also code support:

```
x = 0
x = 2 + 2
what is x
```

Additionally link to Records, AbstractRecords and Selections. You can also **reference** their fields (for example record key)

link to Root (Record) and its key: problem (Record key) looks like this

or like this:

- pause_after_run (Record key)
- problem (Record key)
- SequentialCoupling (Record)
- PartTool (Selection)
- diff_m (Record key)

Using latex in md:

bold and *italic* font

or like this: $x = \frac{1+y}{1+2z^2}$ bigger?

$$x = \frac{1+y}{1+2z^2}$$

or

$$\frac{1}{1+\frac{1}{2+\frac{1}{3+x}}} + \frac{1}{1+\frac{1}{2+\frac{1}{3+x}}}$$

or

$$\begin{aligned} e^x \quad \approx \quad & 1 + x + x^2/2! + \\ & + x^3/3! + x^4/4! + \\ & + x^5/5! \end{aligned}$$

ca asc